

Version 2.0



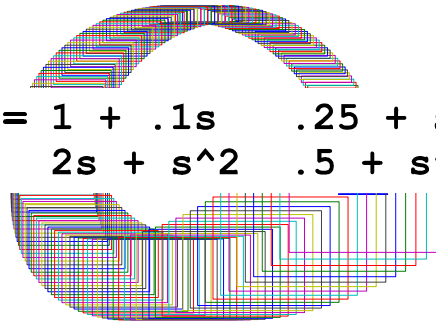
The Polynomial Toolbox for MATLAB

Commands

On-line reference

March, 1999

$$P = \frac{1 + .1s}{2s + s^2} \cdot \frac{.25 + s^3}{.5 + s^2}$$



PolyX, Ltd

E-mail info@polyx.com

Support support@polyx.com

Sales sales@polyx.com

Web www.polyx.com

Tel. +420-2-66052314

Fax +420-2-6884554

Jarni 4

Prague 6, 16000

Czech Republic

Polynomial Toolbox Manual

© COPYRIGHT 1999 by PolyX, Ltd.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from PolyX, Ltd.

Printing history: March 1999. First printing.

Overview

The following tables list all available operations and functions by category.

Table 1. Global structure

gensym	Set global variable symbol for polynomial matrices
gprop	Set/modify global polynomial properties
pformat	Set output format
pinit	Initialize the Polynomial Toolbox
checkpb	Check conflicts of Polynomial Toolbox variables
pver	Polynomial Toolbox version information
pversion	Polynomial Toolbox version number
tolerance	Set global relative tolerance
userdata	Set or return user data of polynomial object
verbose	Set global verbose level

Table 2. Polynomial matrix object properties

deg	Extract various degrees matrices
lcoef	Extract various leading coefficient matrices
lop	Create a polynomial matrix object
pol	Create a polynomial matrix object
pprop	Set/modify properties of polynomial matrix
symbol	Set or return variable symbol of polynomial matrix

Table 3. Convertors

bhf	Converts realization (A,B,C) into upper Hessenberg form
bhf2rmf	Converts realization into a right coprime PMF
dsp2pol	Conversion from DSP format to Polynomial Toolbox format
dss2lmf	Descriptor state space to left PMF
dss2rmf	Descriptor state space to right PMF
dss2ss	Descriptor system to state space system
dssreg	Regularization of a standard descriptor plant
lmf2dss	Left PMF to descriptor state space
lmf2rat	Left PMF to polynomial numerator and denominator matrices
lmf2rmf	Left-to-right conversion of PMF
lmf2ss	LMF to observer form realization (A,B,C,D)
lmf2tf	LMF to Control System Toolbox transfer function
lmf2zpk	LMF to Control System Toolbox zero-pole-gain format
lti2lmf	LTI object to left polynomial matrix fraction
lti2rmf	LTI object to right polynomial matrix fraction
mat2pol	Conversion from MATLAB format to Polynomial Toolbox format
new2old	Conversion to old polynomial matrix format
old2new	Conversion from old polynomial format to new
pol2dsp	Conversion from Polynomial Toolbox format to DSP format
pol2mat	Conversion from Polynomial Toolbox format to MATLAB format
pol2root	Extract zeros and gains of polynomial matrix
rat2lmf	Polynomial numerator and denominator matrices to left PMF
rat2rmf	Polynomial numerator and denominator matrices to right PMF
reverse	Reverse the variable of PMF

rmf2dss	Right PMF to descriptor state space
rmf2lmf	Right to left conversion of PMF
rmf2rat	Right PMF to polynomial numerator and denominator matrices
rmf2ss	RMF to Controller form realization (A,B,C,D)
rmf2tf	RMF to Control System Toolbox transfer function.
rmf2zpk	RMF to Control System Toolbox zero-pole-gain format
root2pol	Construct polynomial matrix from its zeros and gains
ss	LMF or RMF to LTI object in state space form
ss2dss	State space to descriptor state space model
ss2lmf	State space to left matrix fraction conversion
ss2rmf	State space to right matrix fraction conversion
sym	Conversion from polynomial matrix to symbolic format
tf	LMF or RMF to LTI object in transfer function form
tf2lmf	Control System Toolbox transfer function to LMF
tf2rmf	Control System Toolbox transfer function to RMF
zpk	LMF or RMF to LTI object in zero-pole-gain form
zpk2lmf	Zero-pole-gain to left matrix fraction
zpk2rmf	Zero-pole-gain to right matrix fraction

Table 4. Overloaded operations

char	Convert a polynomial object to cell array of strings
ctranspose	Conjugate transposition
(')	
display	Command window display of polynomial matrix
eq	Equality test for polynomial matrices
fliplr	Flip a polynomial matrix in left/right direction
flipud	Flip a polynomial matrix in up/down direction
horzcat ([,])	Horizontal concatenation of polynomial matrices

<code>kron</code>	Kronecker tensor product of polynomial matrices
<code>ldivide (.\)</code>	Left polynomial array divide
<code>minus (-)</code>	Binary subtraction of polynomial matrices
<code>mldivide (\)</code>	Backslash or left polynomial matrix divide
<code>mpower (^)</code>	Matrix power for polynomial matrix
<code>mrdivide (/)</code>	Slash or right polynomial matrix divide
<code>mtimes (*)</code>	Matrix multiplication of polynomial matrices
<code>ne</code>	Inequality test for polynomial matrices
<code>plus (+)</code>	Binary addition of polynomial matrices
<code>power (.^)</code>	Element-wise power for polynomial matrix
<code>rdivide (./)</code>	Right array divide
<code>subsasgn</code>	Subscripted assignment for polynomial matrix
<code>subsref</code>	Subscripted reference for polynomial matrix
<code>times (.*)</code>	Element-wise multiplication
<code>transpose</code> <code>(.')</code>	Matrix transposition.
<code>uminus</code>	Unary minus of polynomial matrix
<code>uplus</code>	Unary plus of polynomial matrix
<code>vertcat ([:])</code>	Vertical concatenation of polynomial matrices

Table 5. Overloaded functions

<code>companion</code>	Block companion matrix
<code>conj</code>	Polynomial matrix complex conjugate
<code>det</code>	Compute determinant of square polynomial matrix
<code>det2d</code>	Determinant of 2-D polynomial matrix
<code>diag</code>	Extract diagonals and create diagonal matrices

<code>imag</code>	Imaginary part of polynomial matrix
<code>inv</code>	Inverse of a polynomial matrix
<code>isempty</code>	True for empty polynomial matrix
<code>isfinite</code>	True for finite elements in polynomial matrix
<code>isinf</code>	True for infinite elements in polynomial matrix
<code>isnan</code>	True for Not-a-Number in polynomial matrix
<code>isprime</code>	True for left or right prime polynomial matrix
<code>isreal</code>	True for real polynomial matrix
<code>length</code>	Length of vector
<code>lu</code>	LU factorization for polynomial matrices
<code>norm</code>	Polynomial matrix norms
<code>null</code>	Null space of a polynomial matrix
<code>pinv</code>	Pseudo-inverse of polynomial matrix
<code>polyval</code>	Evaluate a polynomial matrix
<code>prod</code>	Product of elements of polynomial matrix
<code>rank</code>	Polynomial matrix rank
<code>real</code>	Real part of polynomial matrix
<code>roots</code>	Find polynomial matrix roots
<code>rot90</code>	Rotate polynomial matrix 90 degrees
<code>shift</code>	Shift polynomial matrix
<code>size</code>	Polynomial matrix dimensions
<code>sum</code>	Sum of elements of polynomial matrix
<code>sylv</code>	Create Sylvester matrix of a polynomial matrix
<code>trace</code>	Sum of diagonal elements of a polynomial matrix
<code>tril</code>	Extract lower triangular part of polynomial matrix
<code>triu</code>	Extract upper triangular part of polynomial matrix

Table 6. Basic functions (other than overloaded)

adj	Adjoint of square polynomial matrix
charact	Characteristic vectors of a polynomial matrix
evenpart	Return the even part of a polynomial object
hurwitz	Create Hurwitz matrix of polynomial objects
inertia	Inertia of a polynomial matrix
isfullrank	True if polynomial matrix has full rank
isproper	True if polynomial matrix fraction is proper
issingular	True if polynomial matrix is singular
isstable	True if polynomial matrix is stable
isunimod	True if polynomial matrix is unimodular
kharit	Create Kharitonov polynomials
gram	Gramian of polynomial matrix fraction
h2norm	H2 norm of a polynomial matrix fraction
hinfnorm	H-infinity norm of a polynomial matrix fraction
linvt	Linear transform of variable
longldiv	Long left polynomial matrix division
longrdiv	Long right polynomial matrix division
oddpart	Return the odd part of a polynomial object
polfit	Fit polynomial matrix element-by-element to data
polpart	Polynomial matrix symmetric part extraction
polyder	Derivative of a polynomial matrix
prand	Generates polynomial matrix with random coefficients
ptopex	Extreme polynomials for a polytype of polynomials
pzer	Perform zeroing on a polynomial matrix
scale	Scale a polynomial matrix

Table 7. Advanced operations (other than overloaded)

gld	Greatest left divisor of polynomial matrices
grd	Greatest right divisor of polynomial matrices
ldiv	Left polynomial matrix division
llm	Least left multiple of polynomial matrices
lrm	Least right multiple of polynomial matrices
minbasis	Minimal polynomial basis
rdiv	Right polynomial matrix division
stabint	Stability interval of uncertain polynomial matrices

Table 8. Special matrices

d,p,q,s,v,z	Create simple basic monomials
,zi	
mono	Create monomial matrix (vector) of current global variable

Table 9. Matrix pencil routines

clements	Conversion to Clements standard form
pencan	Conversion to real Kronecker canonical form
plyap	Solution of the pencil equation $AX + YB = C$

Table 10. Numerical routines

cgivens1	Calculates Givens rotation
qzord	Ordered QZ transformation
schurst	Ordered complex Schur decomposition of a matrix

Table 11. Canonical and reduced forms

colred	Column reduced form of a polynomial matrix
diagred	Diagonal reduced form of a polynomial matrix
echelon	Echelon form of a polynomial matrix
hermite	Hermite form of a polynomial matrix
pdg	Diagonalization of a polynomial matrix
rowred	Row reduced form of a polynomial matrix
smith	Smith form of a polynomial object
tri	Triangular or staircase form of a polynomial matrix

Table 12. Control routines

debe	Deadbeat controllers of discrete-time linear systems
dsshinf	H-inf suboptimal compensator for descriptor systems
dssmin	Minimize dimension of pseudo state descriptor system
dssrch	Search for the optimal solution of a descriptor H-infinity problem
mixeds	Solution SISO mixed sensitivity problem
plqg	Polynomial solution of a MIMO LQG problem
pplace	Polynomial pole placement
splqg	Polynomial solution of a SISO LQG problem
stab	Stabilizing controllers of linear systems

Table 13. Equation solvers

axb	Solution of $AX = B$
axbc	Solution of $AXB = C$
axbyc	Solution of $AX + BY = C$
axxab	Solution of $A'X + X'A = B$

axyab	Solution of $A'X + Y'A = B$
axybc	Solution of $AX + YB = C$
xaaxb	Solution of $XA' + AX' = B$
xab	Solution of $XA = B$
xaybc	Solution of $XA + YB = C$

Table 14. Factorizations

fact	Polynomial matrix factor extraction
spscof	Polynomial J -spectral co-factorization
spf	Polynomial spectral factorization

Table 15. SIMULINK

polblock	SIMULINK mdl-file
-----------------	-------------------

Table 16. Visualization

khplot	Plot of Kharitonov rectangles for interval polynomials
pplot	2-D plot of polynomial matrix
pplot3	3-D plot of polynomial matrix
ptopplot	Plot polygonal value sets for polytype of polynomials
zpplot	Plot of zero-pole map

Table 17. Graphic user interface

pme	Polynomial Matrix Editor
------------	--------------------------

Table 18. Demonstrations and help

covf	Covariance function of an ARMA process
demoB	Script file for the demo “Control of a batch process”
demoM	Script file for the demo “Polynomial solution of the SISO mixed sensitivity problem”
minsens	Minimum peak sensitivity
poldesk	Comprehensive hypertext documentation

adj, inv**Purpose**

Adjoint and determinant of a square polynomial matrix

Inverse of a square polynomial matrix

Syntax

```
[adjA,detA] = adj(A[,tol])
```

```
[adjA,detA] = adj(A,'int'[,tol])
```

```
[adjA,detA] = adj(A,'def'[,tol])
```

```
[num,den] = inv(A[,tol])
```

```
[num,den] = inv(A,'int'[,tol])
```

```
[num,den] = inv(A,'def'[,tol])
```

Description

The command

```
[adjA,detA] = adj(A)
```

computes the adjoint and the determinant of a square polynomial matrix A . The adjoint is defined as

$$\text{adj } A(s) = (-1)^{i+j} \det A^{ij}(s)$$

$A^{ij}(s)$ is the matrix that results by omitting the i -th column and j -th row from $A(s)$. If A is non-singular and has size n then

$$A(s)\text{adj } A(s) = I_n \det A$$

so that

$$A^{-1}(s) = \frac{1}{\det A} \text{adj } A(s)$$

The command

```
[adjA,detA] = adj(A,'int')
```

uses fast Fourier transform and interpolation for computing **adjA** and **detA**. This is the default method.

If the command takes the form

```
[adjA,detA] = adj(A,'def')
```

then the algorithm proceeds according to the definition of the adjoint matrix and uses fast Fourier transform and interpolation for computing the subdeterminants.

The optional input parameter **tol** defines a local tolerance for zeroing. Its default value is the global zeroing tolerance. No zeroing is applied if *A* is a constant matrix.

The command

```
[num,den] = inv(A)
```

computes a polynomial matrix **num** and scalar monic polynomial **den** such that **num/den** is the inverse of *A*. The command

```
[num,den] = inv(A,'int')
```

uses fast Fourier transform and interpolation for computing **num** and **den**. This is the default method. The command

```
[num,den] = inv(A,'def')
```

proceeds according to the definition of the adjoint matrix and uses fast Fourier transform and interpolation.

The commands

```
[num,den] = inv(A,tol)
```

```
[num,del] = INV(A,method,tol)
```

work with zeroing specified by the input tolerance **tol**, while **method** is **'int'** or **'def'**. The default value of **tol** is the global zeroing tolerance. No zeroing is applied if **A** is a constant matrix.

Examples

Consider

```
P = [ 1+s    s
      1    s^2 ];
```

Then

```
[adjP,detP] = adj(P)
```

results in

```
adjP =
      s^2      -s
      -1      1 + s

detP =
      -s + s^2 + s^3
```

To check the result we calculate

```
P*adjP
ans =
      -s + s^2 + s^3      0
      0      -s + s^2 + s^3
```

It is believed that the matrix

```
U = [      0      0      0.71
      -0.71      0.71      0
      -0.71      -0.71      0.71*s  ];
```

is unimodular. To check this and to compute the inverse of U we issue the command


```
[num,den] = inv(U)
```

This yields

```
num =
```

```
    0.7s    -0.7    -0.7
```

```
    0.7s     0.7    -0.7
```

```
    1.4      0      0
```

```
Constant polynomial matrix: 1-by-1
```

```
den =
```

```
    1
```

This confirms that U is unimodular with inverse `num`. We check the result:

```
U*num
```

```
Constant polynomial matrix: 3-by-3
```

```
ans =
```

```
    1      0      0
```

```
    0      1      0
```

```
    0      0      1
```

Algorithm

With the option '`int`' the routine uses fast Fourier transform and interpolation for computing the adjoint and the determinant.

With the option '`def`' the algorithm proceeds according to the definition of the adjoint matrix and uses fast Fourier transform and interpolation for computing the subdeterminants.

Diagnostics

The macro displays an error messages if the input matrix is not square or if an unknown option is encountered. A warning message is issued if the interpolation method leads to constant matrices that very nearly singular.

See also

<code>det</code>	determinant of a polynomial matrix
<code>pinv</code>	pseudo-inverse of a polynomial matrix

axb, xab, axbc

Purpose Linear polynomial matrix equation solvers

Syntax

```
X = axb(A,B[,tol])
```

```
X = axb(A,B,degree[,tol])
```

```
X = axb(A,B,'sqz'[,tol])
```

```
X = axb(A,B,degrees[,tol])
```

```
[X,K] = axb(A,B[,tol])
```

```
X = axb(A,B,'sylv'[,tol])
```

```
X = xab(A,B[,tol])
```

```
X = xab(A,B,degree[,tol])
```

```
X = xab(A,B,'sqz'[,tol])
```

```
X = xab(A,B,degrees[,tol])
```

```
[X,K] = xab(A,B[,tol])
```

```
X = xab(A,B,'sylv'[,tol])
```

```

X = axbc(A,B[,tol])
X = axbc(A,B,degree[,tol])
X = axbc(A,B,'sqz'[,tol])
X = axbc(A,B,degrees[,tol])
[X,K] = axbc(A,B[,tol])
X = axbc(A,B,'sylv'[,tol])

```

Description

The command

```
X = axb(A,B)
```

solves the linear polynomial matrix equation $AX = B$, where A and B are given polynomial matrices and X is unknown. If no solution exists then a constant matrix filled with **NaNs** is returned.

The command

```
X = axb(A,B,degree)
```

finds a solution of degree given by the parameter **degree**. If **degree** is not specified then a solution of minimum overall degree is computed by an iterative scheme. If **degree** is negative then the function directly returns a solution whose degree equals a computed upper bound.

The command

```
X = axb(A,B, 'sqz' )
```

seeks a solution X with “squeezed” row degrees. If N is the nullity of A then the degrees of the N last rows are minimized, at the expense of increasing the degrees in the other rows.

If **degrees** is a vector of zeros and ones such that **degrees**(i) = 1 and **degrees**(j) = 0 then the function call

```
X = axb(A,B,degrees)
```

attempts to minimize the degree of the i th row in X , provided the degree of the j th row may increase.

The command

```
[X0,K] = axb(A,B)
```

computes besides the minimal-degree solution x_0 additionally a polynomial matrix K whose columns span the right null-space of A . All the solutions to $AX = B$ may then be parametrized as $X = X_0 + KT$, where T is an arbitrary polynomial matrix parameter of appropriate size.

A local zeroing tolerance may be specified by an additional input argument such as **axb**(**A**,**B**,**tol**). Its default value is the global zeroing tolerance.

Note: The meaning of the last of three input arguments may vary. An integer is interpreted as a desired degree while a non-integer is considered as `tol`. If you really need to employ an integer tolerance then you must use four input arguments! Whenever four input arguments are supplied they are strictly required to be in the order **A,B,tol,degrees**.

The command

```
X = axb(A,B,'sylv')
```

solves the equation through the Sylvester matrix method. This is the default method.

The command

```
X = xab(A,B)
```

solves the (other-sided) linear polynomial matrix equation $XA = B$, where A, B are given polynomial matrices and X is unknown. The macro works analogously to **axb**.

The command

```
X = axbc(A,B,C)
```

solves the two-sided linear polynomial matrix equation $AXB = C$, where A, B, C are given polynomial matrices and X is unknown. The macro works as the macros described above with some differences.

The function call

```
X = axbc(A,B,C, 'sqz')
```

seeks a solution X with “squeezed” entry degrees. If N is the nullity of $\mathbf{kron}(B, A')$ then the degrees of the last N entries of the column vector $X(:)$ are minimized, at the expense of increasing the degrees in the other entries.

If **degrees** is an array of the same size as X such that **degrees**(i, j) = 1 and **degrees**(k, l) = 0 then the function call

```
X = axbc(A,B,C, 'sqz', degrees)
```

attempts to minimize the degree of the (i, j)th entry of X , provided the degree of the (k, l)th entry may increase.

The call

```
[X0,K] = axbc(A,B,C)
```

besides the minimum-degree solution **x0** additionally returns a cell array **K** = {**K1**, **K2**, ..., **Kp**}. The **Ki** are solutions to the homogeneous equation $AXB = 0$. All solutions X to the equation $AXB = C$ may be parametrized as

$$X = X_0 + t_1 K_1 + t_2 K_2 + \cdots + t_p K_p$$

where the t_i are arbitrary scalar polynomials.

Examples

We consider several examples.

Example 1 – Divisibility and factor extraction

The polynomial matrix

```
A = [ 1+s  2*s
      1-s  1  ];
```

is conjectured to be a left divisor of a matrix

```
B = [ 1+s+2*s^2  7*s+s^2
      1          3+s-s^2 ];
```

To verify this conjecture and to extract the factor, if it exists, type

```
X = axb(A,B)
```

```
X =
```

```
1      s
s      3
```

The equation is solvable and hence A is a left divisor of B while X is the factor. Indeed,

```
A*X-B
```

```
Zero polynomial matrix: 2-by-2, degree: -Inf
```



```
ans =
    0    0
    0    0
```

As A is square and invertible, the solution is unique.

If we check whether A is perhaps also a right divisor of B then the reply to the command

```
xab(A,B)
Constant polynomial matrix: 2-by-2
ans =
    NaN    NaN
    NaN    NaN
```

is clearly negative.

Example 2 – Equation with a nonsquare A

Consider the 2×3 matrix

```
A = [ 1    s    s^2
      2  1+s    s  ];
```

and the 2×2 matrix

```
B = [ 1+s^2    s^3
      s^2    1+s^2 ];
```

To solve the equation $AX = B$ for the unknown X just type

```
X = axb(A,B)

X =
      1 + 2s + 3s^2    -s - s^2
     -2 - 5s          1 + 2s
      3              -1 + s
```

To check that X verifies the equation type

```
A*X-B

Zero polynomial matrix: 2-by-2, degree: -Inf

ans =

      0      0
      0      0
```

Example 3 – General solution

The equation considered in Example 2 actually has infinitely many solutions. To retrieve all of them call for the general solution:

```
[X0,K] = axb(A,B)
```

```
X0 =
```

$$\begin{bmatrix} 1 + 2s + 3s^2 & -s - s^2 \\ -2 - 5s & 1 + 2s \\ 3 & -1 + s \end{bmatrix}$$

```
K =
```

$$\begin{bmatrix} s^3 \\ s - 2s^2 \\ -1 + s \end{bmatrix}$$

x_0 is the minimum degree solution but K represents the right kernel of A . Any $X = x_0 + K*T$ with an arbitrary polynomial matrix T is also a solution. For instance,

```
Xother = X0+K*[2 s-2]
```

```
Xother =
```

$$\begin{bmatrix} 1 + 2s + 3s^2 + 2s^3 & -s - s^2 - 2s^3 + s^4 \end{bmatrix}$$

$$-2 - 3s - 4s^2$$

$$1 + 5s^2 - 2s^3$$

$$1 + 2s$$

$$1 - 2s + s^2$$

is a solution. Indeed,

A*Xother-B

Zero polynomial matrix: 2-by-2, degree: -Inf

ans =

$$\begin{matrix} 0 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 0 \end{matrix}$$

Example 4 – Minimal and squeezed solution

The solution x_0 obtained in Example 3 has the least overall degree among all solutions, which is 2. Note that the degree of the last row of X equals the degree of the last row of K , namely 1. This means that there must exist another solution whose last row has degree 0 that possibly has an overall degree higher than the minimal degree 2. Such a solution is called “squeezed,” and may be obtained as

Xsqu = axb(A,B,'sqz')

Xsqu =

$$1 + 2s + 3s^2$$

$$-s - s^2 - s^3$$

$$\begin{array}{rcl} -2 & - & 5s \\ & & 1 + s + 2s^2 \\ 3 & & 0 \end{array}$$

Note that the degree of the last row degree has indeed decreased while the overall degree now is 3.

Example 5 – Visualizing the solution process

The default solution returned by `axb` possesses a minimum overall degree of X . Such a particular solution is, in fact, found in a series of steps by searching for particular solutions within a certain range of degrees. The whole process may be displayed by setting the *verbose level* equal to 'yes'.

```
verbose yes
```

```
X = axb(A,B);
```

```
XAB: Seek minimum degree solution.
```

```
XAB: Attempt # 1 Degree 1 (Min 1 / Max 5) No.
```

```
XAB: Attempt # 2 Degree 3 (Min 1 / Max 5) Yes.
```

```
XAB: Attempt # 3 Degree 2 (Min 1 / Max 3) Yes.
```

```
XAB: Solve equation with QR factorization.
```

```
XAB: A solution of degree 2 was found.
```

The solution equals that found before.

Example 6 – Solution of a prescribed degree

For large matrices of high degree one may wish to shorten the search for the minimum degree solution. If the desired degree is supplied as input argument then the macro looks just for a solution of the particular desired degree and then stops regardless of what has been found. Thus,

```
X = axb(A,B,1)
```

results in

```
XAB: Seek solution of degree 1.
```

```
XAB: Attempt # 1 Degree 1 (Min 1 / Max 5) No.
```

```
XAB: No polynomial solution was found.
```

```
Constant polynomial matrix: 3-by-2
```

```
X =
```

```
NaN      NaN  
NaN      NaN  
NaN      NaN
```

If a negative desired degree is entered then the upper bound on the degree bound is automatically chosen. This is useful as a fast check for solvability. Thus,

```
X = axb(A,B,-1)
```

confirms that the equation is solvable:

```
XAB: Seek solution of degree 5.
```

```
XAB: Attempt # 1 Degree 5 (Min 5 / Max 5) Yes.
```

```
XAB: Solve equation with SVD.
```

```
XAB: A solution of degree 5 was found.
```

```
Polynomial matrix in s: 3-by-2, degree: 5
```

```
X =
```

```
Column 1
```

```
1 + 2s + 3s^2 + 1.2s^3 + 0.52s^4 + 0.2s^5  
-2 - 3.8s - 1.9s^2 - 0.85s^3 - 0.39s^4  
1.8 + 0.67s + 0.33s^2 + 0.2s^3
```

```
Column 2
```

```
-s - s^2 - 0.53s^3 - 0.087s^4 - 0.033s^5
```

$$1 + 1.5s + 0.98s^2 + 0.14s^3 + 0.065s^4$$

$$-0.47 + 0.55s - 0.054s^2 - 0.033s^3$$

Algorithm

For given degrees of the entries of the unknown polynomial matrix X Kronecker products are used to expand the polynomial equation that needs to be solved into a (large) set of linear equations in the unknown coefficients. The required degrees are found by a systematic search depending on the options that are specified.

Diagnostics

The macro displays an error messages if

- The second input argument is missing
- Invalid **Not-a-Number** or **Infinite** entries are found in the input matrices
- Inconsistent variables are encountered in the input matrices
- The input matrices have inconsistent dimensions
- An invalid option is encountered
- The dimension of the degree index vector is wrong
- The dimension of the polynomial null space is incorrect

See also

axbyc, **axybc**, **xaybc** solution of various types of Diophantine equations
axxab, **axyab**, **xaaxb** symmetric and asymmetric equation solvers

axbyc, axybc, xaybc

Purpose Diophantine equation solvers

Syntax

```
[X,Y] = axbyc(A,B,C[,tol])
[X,Y] = axbyc(A,B,C,degree[,tol])
[X,Y] = axbyc(A,B,C,'minx'[,tol])
[X,Y] = axbyc(A,B,C,'miny'[,tol])
[X,Y,R,S] = axbyc(A,B,C[,tol])
[X,Y] = axbyc(A,B,C,'syl'[,tol])

[X,Y] = axybc(A,B,C[,tol])
[X,Y] = axybc(A,B,C,degree[,tol])
[X,Y] = axybc(A,B,C,'minx'[,tol])
[X,Y] = axybc(A,B,C,'miny'[,tol])
[X,Y,R,S] = axybc(A,B,C[,tol])
[X,Y] = axybc(A,B,C,'syl'[,tol])
```

```
[X,Y] = xaybc(A,B,C[,tol])
```

```
[X,Y] = xaybc(A,B,C,degree[,tol])
```

```
[X,Y] = xaybc(A,B,C,'minx'[,tol])
```

```
[X,Y] = xaybc(A,B,C,'miny'[,tol])
```

```
[X,Y,R,S] = xaybc(A,B,C[,tol])
```

```
[X,Y] = xaybc(A,B,C,'sylv'[,tol])
```

Description

Diophantine equations with polynomial matrices are a special type of linear equation. They often arise in control theory and, hence, deserve special attention. Typically, the polynomial matrix A is square invertible being the “denominator” of some transfer matrix.

The command

```
[X,Y] = axbyc(A,B,C)
```

solves the linear polynomial matrix equation $AX + BY = C$, where A , B and C are given polynomial matrices and X and Y are unknown. If no solution exists then constant matrices X and Y filled with **NaNs** are returned.

The command

```
[X,Y] = axbyc(A,B,C,degree)
```

finds a solution such that the degree of the composite matrix $[X; Y]$ is given by the parameter **degree**. If **degree** is not specified then a solution of minimum overall degree is computed by an iterative scheme. If **degree** is negative then the function directly returns a solution whose degree equals a computed upper bound.

The commands

```
[X,Y] = axbyc(A,B,C,'minx')
```

```
[X,Y] = axbyc(A,B,C,'miny')
```

determine a solution (X, Y) with minimum degree of X or Y , respectively.

The command

```
[X0,Y0,R,S] = axbyc(A,B,C)
```

computes besides the minimal-degree solution $(\mathbf{x0}, \mathbf{y0})$ additionally the right null space of $[A \ B]$, so that all solutions to $AX + BY = C$ may be parametrized as

$$X = \mathbf{x0} + RT$$

$$Y = \mathbf{y0} + ST$$

T is an arbitrary polynomial matrix parameter of appropriate dimensions.

A local zeroing tolerance may be specified by an additional input argument such as **axbyc(A,B,C,tol)**. Its default value is the global zeroing tolerance.

Note: The meaning of the last of four input arguments may vary. An integer is interpreted as a desired degree while a non-integer is considered as `tol`. If you really need to employ an integer tolerance then you must use five input arguments! Whenever five input arguments are supplied they are strictly required to be in the order **A,B,C,tol,degrees**.

The command

```
[X,Y] = axbyc(A,B,C,'sylv')
```

solves the equation through the Sylvester matrix method. This is the default method.

The command

```
[X,Y] = xabyc(A,B,C)
```

solves the (other-sided) linear polynomial matrix equation $XA + BY = C$, where A , B and C are given polynomial matrices and X and Y are unknown. The macro works analogously to **axbyc**.

The command

```
X = axybc(A,B,C)
```

solves the two-sided linear polynomial matrix equation $AX + YB = C$, where A , B , C are given polynomial matrices and X and Y are unknown. The macro works as the macros described above with some differences.

The function call

[X,Y] = axybc(A,B,C,'min')

seeks a solution such that the degrees of the N last entries of the column vector $Z = [X(:); Y(:)]$ are minimized, where N is the nullity of the matrix $[\text{krón}(I, A) \text{krón}(B, I)]$.

When **degrees** is a vector of the same length as Z such that **degrees**(i) = 1 and **degrees**(j) = 0 then the function call

[X,Y] = axybc(A,B,C,'min',degrees)

attempts to minimize the i th entry degree in Z provided the j th entry degree may increase.

The command

[X0,Y0,R,S] = AXYBC(A,B,C)

returns besides the minimum degree solution (**X0**, **Y0**) also two cell arrays **R** = {**R1**, **R2**, ...} and **S** = {**S1**, **S2**, ...} such that the **Ri** and **Si** are solutions to the homogeneous equation $AR + SB = 0$. All solutions to equation $AX + YB = C$ may be parametrized as

$$X = X_0 + t_1 R_1 + t_2 R_2 + \dots + t_p R_p$$

$$Y = Y_0 + t_1 S_1 + t_2 S_2 + \dots + t_p S_p$$

where the t_i are arbitrary scalar polynomials.

Examples

Example 1 – Scalar Diophantine equation

Consider three scalar polynomials

$$a = -1 + z^2;$$

$$b = z;$$

$$c = 1 - 2z;$$

The scalar Diophantine equation may straightforwardly be solved by the command

$$[x, y] = \text{axbyc}(a, b, c)$$

Constant polynomial matrix: 1-by-1

$$x =$$

$$-1$$

$$y =$$

$$-2 + z$$

The same result is obtained by the other macros:

$$[x, y] = \text{xaybc}(a, b, c)$$

Constant polynomial matrix: 1-by-1

$$x =$$

```

-1
y =
-2 + z
[x,y] = axybc(a,b,c)
Constant polynomial matrix: 1-by-1
x =
-1
y =
-2 + z

```

All solutions of the equation may be constructed with the help of the command

```

[x0,y0,r,s] = axbyc(a,b,c)
Constant polynomial matrix: 1-by-1
x0 =
-1
y0 =
-2 + z

```

```

r =
    -z

s =
    -1 + z^2

```

Choosing the arbitrary parameter t equal to 1 we obtain

```

t = 1; x = x0+r*t, y = y0+s*t

x =
    -1 - z

y =
    -3 + z + z^2

```

Example 2 – Solutions of minimum degree in X or Y

By default, the Diophantine equation solvers return a solution of minimal overall degree of $[X; Y]$. If the degree of the right hand side C is high then one may further minimize the degree of either X or Y while possibly increasing the degree of the other. Consider

```
clear, gprop s, a = 1+s; b = s; c = 1-s^2;
```

and solve


```
[x,y] = axbyc(a,b,c)
```

```
x =
```

```
1 - 0.67s
```

```
y =
```

```
-0.33 - 0.33s
```

Besides this default solution with minimum overall degree 1 there also exists a solution with lower and indeed minimal degree of y :

```
[x1,y1] = axbyc(a,b,c,'miny')
```

```
x1 =
```

```
1 - s
```

```
Zero polynomial matrix: 1-by-1, degree: -Inf
```

```
y1 =
```

```
0
```

In turn there also is a solution with minimum degree of x :

```
[x2,y2] = axbyc(a,b,c,'minx')
```

```
Constant polynomial matrix: 1-by-1
```

```

x2 =
    1
y2 =
    -1 - s

```

Example 3 – Matrix Diophantine equation

Consider the matrices

```

clear, gprop z
A = [ z^2    0
      0      z ];
B = [ 1      0
      z      0
      0      1 ];
C = [ z^2   -1
      0      z ];

```

and solve the one-sided equation

```
[X,Y] = xaybc(A,B,C)
```

Constant polynomial matrix: 2-by-2

X =

1	0
0	1

Constant polynomial matrix: 2-by-3

Y =

0	0	-1
0	0	0

To obtain a general solution, type

`[X0,Y0,R,S] = xaybc(A,B,C)`

Constant polynomial matrix: 2-by-2

X0 =

1	0
0	1

Constant polynomial matrix: 2-by-3

Y0 =

$$\begin{array}{ccc} 0 & 0 & -1 \end{array}$$

$$\begin{array}{ccc} 0 & 0 & 0 \end{array}$$

Constant polynomial matrix: 3-by-2

R =

$$\begin{array}{cc} 0 & 0 \end{array}$$

$$\begin{array}{cc} -1 & 0 \end{array}$$

$$\begin{array}{cc} 0 & -1 \end{array}$$

S =

$$\begin{array}{ccc} z & -1 & 0 \end{array}$$

$$\begin{array}{ccc} 0 & z & 0 \end{array}$$

$$\begin{array}{ccc} 0 & 0 & z \end{array}$$

All solutions are given by

$$X = \mathbf{x}0 + TR$$

$$Y = \mathbf{y}0 + TS$$

with T an arbitrary 2×3 polynomial matrix.

Algorithm

After suitable preparation the macros call the routines **xab** or **axbc**.

Diagnostics

The macros display error messages if

- not enough input arguments are provided
- **Not-a-Number** or **Infinite** entries are encountered in the input matrices
- the input matrices have inconsistent dimensions

See also

axb, **xab**, **axbc** linear polynomial matrix equations

axxab, **axyab**, **xaaxb** symmetric and asymmetric equation solvers

axxab, axyab, xaaxb

Purpose Symmetric and asymmetric polynomial equation solvers

Syntax

```

X = axxab(A,B[,tol])

X = axxab(A,B,'syl'[,tol])

X = axxab(A,B,'tri'[,tol])

X = axxab(A,B,coldeg[,tol])

X = axxab(A,B,'red'[,tol])

X = xaaxb(A,B[,tol])

X = xaaxb(A,B,'syl'[,tol])

X = xaaxb(A,B,'tri'[,tol])

X = xaaxb(A,B,rowdeg[,tol])

X = xaaxb(A,B,'red'[,tol])

[X,Y] = axyab(A,BL,BR[,tol])

```

```
[X,Y] = axyab(A,BL,BR,'sylv',[ ,tol])
```

```
[X,Y] = axyab(A,BL,BR,'red',[ ,tol])
```

Description

The command

```
X = axxab(A,B)
```

solves the bilateral symmetric matrix polynomial equation

$$A'X + X'A = B$$

where B is a para-Hermitian polynomial matrix.

In the continuous-time case, that is, if the variable of A and B is s (or p), then B is para-Hermitian if

$$B(s) = B^T(-s)$$

and B is provided to **axxab** as it is.

In the discrete-time case, that is, if the variable of A and B is z (or q , d or z^{-1}) then B is para-Hermitian if

$$B(z) = B^T(1/z)$$

The matrix B is not a polynomial matrix (except in the trivial case that it is a constant matrix) but has the form

$$B(z) = B_d^T z^{-d} + B_{d-1}^T z^{-d+1} + \cdots + B_1^T z^{-1} + B_0 + B_1 z + \cdots + B_d z^d$$

It needs to be supplied to the macro **axxab** in the polynomial form

$$B(z) := z^d B(z)$$

The degree offset d is evaluated upon cancellation of the leading and trailing zero matrix coefficients. If there are no zero coefficients then the degree offset is $\deg(B/2)$.

The command

```
X = axxab(A,B,'sylv')
```

solves the equation with the Sylvester matrix method. This is the default method. It can be used with the following modifiers:

- In the form

```
X = axxab(A,B,'tri')
```

the macro returns a solution with upper-triangular columnwise leading coefficient matrix (in the continuous-time case) or upper-triangular absolute coefficient matrix (in the discrete-time case).

- In the continuous-time case the command

```
X = axxab(A,B,coldeg)
```

computes a solution X with column degrees **coldeg**. By default,

$$\text{coldeg}(i) = \max(\deg(B) - \deg(A), 0)$$

for all column indices i . This option is not available for the discrete-time case, where the macro always computes a solution X of degree

$$\deg(X) = \max(\deg(A), \deg(B)).$$

The command

X = axxab(A,B,'red')

solves the equation with polynomial reductions, which is a version of the Euclidean division algorithm for polynomials. In the continuous-time case the macro computes a solution with upper-triangular columnwise leading coefficient and such that XA^{-1} is proper under the assumptions that A is stable and

$$(A^{-1})^T B A^{-1}$$

is biproper. In the discrete-time case the macro computes a solution X such that its zero coefficient matrix is upper-triangular under the assumption that A is stable.

If there is no solution of the specified degree then all the entries in X are set equal to **NaN**.

An optional tolerance **tol** may be specified as an additional input argument. Its default value is the global zeroing tolerance.

The command

X = xaaxb(A,B)

works very similarly to `axxab` except that it solves the symmetric equation

$$XA' + AX' = B$$

with B para-Hermitian.

The command

$$[X, Y] = \text{axyab}(A, BL, BR)$$

solves the scalar polynomial equation

$$A'X + Y'A = BL' + BR$$

for given polynomials A , BL and BR . A needs to be stable. In the discrete-time case a solution always exists and if in the continuous-time case

$$\max(\deg(BL), \deg(BR)) \leq \deg(A)$$

then a solution always exists and the polynomials X and Y are such that $\max(\deg(X), \deg(Y)) \leq \deg(A)$. The command

$$[X, Y] = \text{axyab}(A, BL, BR, 'syl')$$

uses the Sylvester matrix algorithm. This is the default method. The algorithm of

$$[X, Y] = \text{axyab}(A, BL, BR, 'red')$$

uses the polynomial reduction algorithm, which is a version of the Euclidean division algorithm for polynomials.

Examples

Let

```

A = [ -8+s      1
      6+6*s     1 ];

B = [ 100-37*s^2  -2-7*s
      -2+7*s      2      ];

```

Then

```
X = axxab(A,B)
```

yields

```

X =
      -4 + 0.5s      0.5
      3 + 3s      0.5

```

In the version

```
X = axxab(A,B,'tri')
```

the result is

```

X =
      -3.9 + 18s      0.51 - 1.7s

```

3.1

0.49 + 0.28s

Next consider the asymmetric scalar discrete-time equation

$$\underbrace{(2+z^{-1})}_{A'(z)}X(z) + Y'(z)\underbrace{(2+z)}_{A(z)} = \underbrace{4z^{-2}+9z^{-1}+1}_{BL'(z)} + \underbrace{1+7z+3z^2}_{BR(z)}$$

It may be solved with the commands

```
A = 2+z; BL = 4+9*z+z^2; BR = 1+7*z+3*z^2;
```

```
[X,Y] = axyab(A,BL,BR)
```

This yields

```
X =
```

```
-1.2 + 2.8z + 1.5z^2
```

```
Y =
```

```
-0.17 + 4.8z + 0.5z^2
```

Algorithm

The Sylvester algorithm for the continuous-time symmetric matrix case is described in Henrion and Sebek (1998c) and for the discrete-time case in Henrion and Sebek (1998d). By using Kronecker products and equating the coefficients of like powers of a set of linear equations is obtained and solved for the unknown coefficients. The solution of the asymmetric scalar equation follows similarly (Stefanidis et al., 1992).

Jezek and Kucera (1985) describe the reduction algorithm. In the symmetric matrix case, the equation is postmultiplied by the adjoint of A and premultiplied by the transpose of this adjoint. This results in a set of scalar symmetric polynomial equations, which are solved by the Euclidean algorithm. By successive substitutions (the forward path of the Euclidean division algorithm) the degrees of the polynomials are decreased until zero. Then the constant scalar problem is directly solved and by performing backward substitutions a solution of the original equation is found. The solution of the scalar asymmetric equation by the reduction method follows similarly.

Diagnostics

The macros **axxab** and **xaaxb** issue error messages under the following conditions:

- The second input argument is missing
- Invalid **Not-a-Number** or **Inf** entries are found in the input matrices
- The input matrices have inconsistent dimensions
- An invalid option or input argument is encountered
- Various assumptions on the input are not satisfied
- The algorithm fails in one of several ways

The macro **axyab** displays an error message if

- The number of input arguments is invalid

- The input polynomials have inconsistent variables
- Invalid **Not-a-Number** or **Inf** entries occur in the input polynomials
- The input arguments are not scalar polynomials
- An invalid option or input argument is encountered
- The first input polynomial is not strictly stable

See also

`axb`, `axbc`, `xab`, linear polynomial equation solvers
`axbyc`, `axybc`, `xaybc`

bhf, bhf2rmf**Purpose**

Convert the controllable part of a state space realization to upper block Hessenberg form.

Obtain a right coprime polynomial matrix fraction description of a minimal state space realization in upper block Hessenberg form.

Syntax

```
[F,G,H,bsizes] = bhf(A,B,C[,tol])
```

```
[N,D] = bhf2rmf(F,G,H,bsizes)
```

Description

The function call

```
[F,G,H,bsizes] = bhf(A,B,C)
```

converts a state space realization $\dot{x} = Ax + Bu$, $y = Cx$, into upper block Hessenberg form. The controllable part of the system (A, B, C) is returned as the system (F, G, H) , where F is an upper block Hessenberg matrix. G has nonzero elements in its first m rows only, where m is the rank of B . The vector **bsizes** contains the sizes of the different blocks of the matrix F . The optional input parameter **tol** is a tolerance that is used in determining the block sizes.

The function

```
[N,D] = bhf2rmf(F,G,H,bsizes)
```

constructs a right coprime polynomial matrix fraction representation of a minimal state space realization in upper block Hessenberg form.

The routines are called by `ss2rmf` and `ss2lmf`.

Examples

Let

```
A = [ 0 1 4 0 0 0 0 0
      3 0 0 0 0 0 0 1
      0 0 3 0 0 0 0 1
      4 0 0 1 0 0 0 0
      0 0 0 0 1 0 0 0
      0 0 0 0 0 0 0 0
      0 0 0 1 0 1 0 0
      1 0 0 0 1 0 0 1 ];

B = [ 1 1 0
      0 0 0
      0 0 1
      0 0 0
      0 0 0
      0 0 0
      0 0 0
```



```

0 0 0
0 1 0 ];
C = [ 0 0 0 0 0 0 0 1 1 ];

```

The command

```
[F,G,H,bsizes] = bhf(A,B,C)
```

returns

```

F =
    3.0000   -1.0000    0.0000   -0.0000    0.0000   -
    0.0000
   -0.0000    1.0000   -1.0000    0.0000    0.0000   -
    0.0000
    4.0000   -0.0000    0.0000   -1.0000         0
    0.0000
    0.0000    1.0000   -3.0000    0.0000    0.0000   -
    0.0000
    0.0000   -0.0000    4.0000         0    1.0000   -
    0.0000

```

```

-0.0000  -0.0000  0.0000  -0.0000  -1.0000  -
0.0000
G =
    0.0000  -0.0000  1.0000
    0.0000  -1.0000  -0.0000
    1.0000   1.0000  0.0000
         0  -0.0000         0
         0         0         0
         0   0.0000         0

H =
    -0.0000  -1.0000  0.0000  -0.0000         0  -
1.0000
bsizes =
      3      2      1

```

This minimal state space realization in upper block Hessenberg form may be converted into a right coprime matrix fraction by the command

```
[N,D] = bhf2rmf(F,G,H,bsizes)
```

It yields

$N =$

$$-1 - 0.75s + 0.75s^2 \quad -s \quad 0$$

$D =$

$$-0.5s + 1.5s^2 - s^3 \quad 1 - s + s^2 \quad -4$$

$$0.5s - 1.3s^2 + 0.75s^3 \quad s - s^2 \quad 0$$

$$0.75s - 0.75s^2 \quad s \quad -3 + s$$

Algorithm

The algorithm for **bhf** and **bhf2rmf** is described in Srijbos (1995, 1996). Using a sequence of Householder transformations (see **hh1** and **hhr**) the matrix A is brought into block Hessenberg form and the uncontrollable part of the system is removed.

Diagnostics

The macros display no error messages

See also

ss2lmf, **ss2rmf** conversion from state space representation to left and right matrix fraction representation

cgivens1

Purpose

Complex Givens rotation

Syntax

`[c,s] = cgivens1(x,y)`

Description

Given two complex numbers x and y the function

`[c,s] = cgivens1(x,y)`

computes a real number c and a complex number s such that the matrix

$$\begin{bmatrix} c & s \\ -s' & c \end{bmatrix}$$

is unitary and

$$\begin{bmatrix} c & s \\ -s' & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} z \\ 0 \end{bmatrix}$$

Here z is an arbitrary complex number.

Examples

First we consider

```
[c,s] = cgivens1(1,i)
```

```
c =
```

```
0.7071
```

```
s =  
0 - 0.7071i
```

Next

```
[c,s] = cgivens1(0,0)  
  
c =  
1  
  
s =  
0
```

Algorithm

The numbers c and s are computed as follows.

- If x is nonzero then

$$c = \frac{|x|}{\sqrt{|x|^2 + |y|^2}}, \quad s = \frac{cy'}{x'}$$

- If $x = 0$ and y is nonzero then $c = 0$ and $s = 1$.
- If $x = 0$ and $y = 0$ then the Givens rotation is not well-defined. The μ -Tools MATLAB toolbox implementation of the complex Givens rotation chooses $c = 0$ and $s = 1$. A better choice is $c = 1$ and $s = 0$.

Diagnostics

The macro displays no error or warning messages.

See also

`qzord` Ordered QZ transformation

char

Purpose Convert a polynomial object to strings

Syntax

```
str = char(P)

str = char(P,N)
```

Description The command

```
str = char(P,N)
```

converts the scalar polynomial P to a string `str` where each coefficient is expressed with N significant digits.

The default

```
char(P)
```

is similar to `char(P,4)` but with a precision of roughly 4 digits and an exponent if required.

If P is a polynomial matrix then the result is a cell array of the same dimension as P consisting of strings that correspond to the entries of P . Each scalar coefficient is taken with a precision of at most N digits.

The command

```
char(P,format)
```

works like `char(P,N)` but uses the format string `format` for each scalar coefficient (see `sprintf` for details). The command

```
char(P,'rat')
```

is a special case of `char(P,format)` and uses rational approximations of the coefficients.

Examples

The command

```
char(1+4/3*s)
```

returns

```
ans =  
1 + 1.3333s
```

while the command

```
char(1+4/3*s,8)
```

yields

```
ans =  
1 + 1.3333333s
```

On the other hand,

```
char(1+4/3*s,'rat')
```

results in


```
ans =  
1 + 4/3*s
```

Finally,

```
char([1+2*s 3*s^2])
```

yields

```
ans =  
' 1 + 2s'      ' 3s^2'
```

Algorithm

The macro uses standard MATLAB 5 operations.

Diagnostics

The macro displays an error message if the macro does not have enough input arguments.

See also

`display` display a polynomial matrix

charact

Purpose Characteristic vectors of a polynomial matrix

Syntax `[V,n,g] = charact(P,z[,tol])`

Description Given a square polynomial matrix P and a complex number z , the command

`[V,n,g] = charact(P,z)`

returns a cell array V of generalized right characteristic vectors of P associated with the zero z . The zero z has algebraic multiplicity n and geometric multiplicity g .

If z has algebraic multiplicity $n = m_1 + m_2 + \dots + m_g$ then there are n nonzero generalized characteristic vectors $V_{\{1,1:m_1\}}$, $V_{\{2,1:m_2\}}$..., $V_{\{g,1:m_g\}}$. The vectors are ordered into g chains $V_{\{i, : \}}$, $i = 1, 2, \dots, g$, each of length m_i and such that $m_1 \geq m_2 \geq \dots \geq m_g$. The first vectors in each chain, that is, the vectors $V_{\{i,1\}}$, $i = 1, 2, \dots, g$, are linearly independent.

Let $P[k]$ denote the k th derivative of P evaluated at z . Then the characteristic vectors in chain number i satisfy

$$0 = P[0]*V_{\{i,1\}}$$

$$0 = P[0]*V_{\{i,2\}} + P[1]*V_{\{i,1\}}$$

...

$$0 = P[0]*V\{i,mi\} + P[1]*V\{i,mi-1\} + \dots + P[mi-1]*V\{i,1\}/(mi-1)!$$

If z is not a zero of P then V is empty and $n = g = 0$.

A tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Consider

$$P = \begin{bmatrix} -1+s & -2+2*s \\ 3-6*s+3*s^2 & 4-8*s+4*s^2 \end{bmatrix};$$

We first compute the roots of P according to

```
roots(P)
```

This yields

```
ans =  
  
1.0000  
  
1.0000 + 0.0000i  
  
1.0000 - 0.0000i
```

It looks as if there is a triple root at 1. Typing

```
[V,n,g] = charact(P,1)
```

results in

```
V =
    [2x1 double]    [2x1 double]
    [2x1 double]    []

n =
    3

g =
    2
```

Apparently this root has algebraic multiplicity 3 and geometric multiplicity 2. The two chains of generalized eigenvectors are retrieved as

```
v{1,:}
ans =
    -2
    1

ans =
    0
```

```

1
and
v{2,:}
ans =
1
0
ans =
[]

```

Algorithm

The algorithm is based upon successive extractions by `cef` and `nullref` of a basis in column echelon form for the right null-spaces of the constant Toeplitz matrices

$$\begin{aligned}
 T_0 &= P[0] \\
 T_1 &= \begin{bmatrix} P[0] & P[1] \\ 0 & P[0] \end{bmatrix} \\
 T_2 &= \begin{bmatrix} P[0] & P[1] & P[2]/2! \\ 0 & P[0] & P[1] \\ 0 & 0 & P[0] \end{bmatrix} \\
 T_3 &= \dots\dots
 \end{aligned}$$

$P[k]$ denotes the k th order derivative of $P(s)$ evaluated at the zero z . For a detailed description of the algorithm see Henrion and Sebek (1998e).

Diagnostics

The macro issues error messages under the following conditions:

- The input matrix is not square
- The second input argument is missing
- An invalid second input argument is found
- The input matrix is singular
- An invalid zero characteristic vector is obtained

See also

fact polynomial matrix factor extraction

roots roots of a polynomial matrix

checkpb, pver, pversion

Purpose	Check conflicts of Polynomial Toolbox variables with existing variables Polynomial Toolbox version information Polynomial Toolbox version number
Syntax	<code>T = checkpb</code> <code>checkpb</code> <code>pver</code> <code>pversion</code>
Description	The command <code>T = checkpb</code> returns <ul style="list-style-type: none">• <code>T = []</code> if the Polynomial Toolbox is set correctly• <code>T = 1</code> if the Polynomial Toolbox is not initialized or the global property structure is incorrect• <code>T = 2</code> if the global property <code>variable</code> has an incorrect value

- **T** = 3 if the global property **tolerance** has an incorrect value
- **T** = 4 if the global property **verbose** has an incorrect value
- **T** = 5 if the global property **format** has an incorrect value
- **T** = 6 if any of the variables *s*, *p*, *zi*, *d*, *z*, *q* or *v* already exists and the polynomial function of the same name is not active

If there are several problems at the same time then the macro returns a vector of “error codes” as listed. If **checkpb** is called without output argument then it displays a status message.

The command

pver

displays the current Polynomial Toolbox version number.

The call

pversion

returns a string containing the Polynomial Toolbox version number, while

[V,D] = pversion

also returns the version release date.

Examples

After typing

```
clear all
```

the command

```
checkpb
```

results in

```
*Global property structure incorrect. Use PINIT!
```

The situation is corrected by typing

```
pinit; checkpb
```

```
Polynomial Toolbox initialized. To get started, type one of  
these: helpwin or poldesk. For product information, visit  
www.polyx.com or www.polyx.cz.
```

```
*Polynomial Toolbox is correctly set.
```

The default definition of the indeterminate variables is disturbed by a command such as

```
s = 1;
```

Typing

```
checkpb
```

now produces the message

```
*Polynomial function s is not active.
```

```
Clear or rename the variable s to activate the function.
```

From the command

```
pver
```

```
-----  
POLYNOMIAL TOOLBOX  Version 2.0 Beta on PCWIN  
-----
```

we obtain the version information, while

```
pversion
```

```
ans =
```

```
2.0 Beta
```

displays the version number only.

Algorithm

The macros use standard MATLAB 5 commands.

Diagnostics

The macros return error messages if the input is inappropriate.

See also

`pinit`

Initialize the Polynomial Toolbox

clements

Purpose Transformation of a para-Hermitian pencil to Clements form

Syntax

```
[C,u,p] = clements(P)
[C,u,p] = clements(P,q)
[C,u,p] = clements(P,q,tol)
```

Description The command

```
[C,u,p] = clements(P,q,tol)
```

transforms the para-Hermitian nonsingular real pencil $P(s) = sE + A$ to Clements standard form C according to

$$C(s) = u(sE + A)u^T = se + a$$

The pencil $sE + A$ is assumed to have no finite roots on the imaginary axis.

The matrix u is orthogonal. The pencil C has the form

$$C(s) = se + a = \begin{bmatrix} 0 & 0 & se_1 + a_1 \\ 0 & a_2 & se_3 + a_3 \\ -se_1^T + a_1^T & -se_3^T + a_3^T & se_4 + a_4 \end{bmatrix}$$

The pencil $se_1 + a_1$ has size $p \times p$ and its finite roots have strictly positive real parts. The matrix a_2 is diagonal with the diagonal entries in order of increasing value.

If the optional input argument q is not present then a_2 has the largest possible size. If q is present and the largest possible size of a_2 is greater than $q \times q$ then – if possible – the size of a_2 is reduced to $q \times q$. Setting $q = \text{Inf}$ has the same effect as omitting the second input argument.

The optional input parameter $\text{tol} = [\text{tol1} \text{ tol2}]$ defines two tolerances. The tolerance tol1 is used in determining whether a generalized eigenvalue of the pencil is infinite. Its default value is $1\text{e-}12$. The tolerance tol2 is used to test whether the pencil has roots on the imaginary axis. It also has the default value $1\text{e-}12$.

If the pencil has eigenvalues on the imaginary axis then the function returns $p = -\text{Inf}$, $e = E$, $a = A$, and u the unit matrix.

Example

We consider the computation of the Clements form of the para-Hermitian pencil

$$P(s) = \begin{bmatrix} 100 & -0.01 & s & 0 \\ -0.01 & -0.01 & 0 & 1 \\ -s & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

We first input this matrix as

```
P = [100 -0.01 s 0; -0.01 -0.01 0 1; -s 0 -1 0; 0 1 0 0];
```

and next compute its Clements form:

```
[C,u,p] = clements(P);
```

We have

```
p, C = pzer(C)
```

```
p =
```

```
1
```

```
C =
```

```

      0      0      0      -10 + s
      0     -1.005      0     -3.5e-005-
0.0007s
      0      0      0.9950     -0.014+0.00071s
     -10 - s    -3.5e-005+0.0007s    -0.014-0.00071s    99

```

We see that

$$se_1 + a_1 = s - 10, \quad a_2 = \begin{bmatrix} -1.005 & 0 \\ 0 & 0.9950 \end{bmatrix}$$

Next we attempt to reduce a_2 to the smallest possible size:

```
[C,u,p] = clements(P,0);
```

```
p, C = pzer(C)
```

```
p =
```

```
2
```

```
Polynomial matrix in s: 4-by-4, degree: 1
```

```
C =
```

```
Columns 1 through 3
```

```
0      0      0
```

```
0      0      1
```

```
0      1     -0.01
```

```
-10-s    -0.01-4.9751e-006s  -0.0099005-0.00099502s
```

```
Column 4
```

```
-10 + s
```

```
-0.01 + 4.9751e-006s
```

```
-0.0099005 + 0.00099502s
```

```
99
```

We now have

$$se_1 + a_1 = \begin{bmatrix} 0 & s - 10 \\ 1 & 4.9751 \times 10^{-6} s - 0.01 \end{bmatrix}$$

while a_2 is the empty matrix.

Algorithm

The algorithm is described in Clements (1993) and in slightly more detail in Kwakernaak (1998).

Diagnostics

The macro displays error messages in the following situations:

- The input matrix is not a square pencil
- The input matrix is not real
- The input pencil is not para-Hermitian
- The input parameter `tol` is not a 1×2 vector

A warning message is issued if the relative residue exceeds $1e-6$. The “relative residue” is the norm of the juxtaposition of the (1,1) and (1,2) blocks of C divided by the norm of P . If verbose mode is switched on then a warning is issued if the pencil has roots on the imaginary axis and the relative residue is reported.

See also

`dsshinf` H_∞ -suboptimal compensators for descriptor systems

colred, diagred, rowred

Purpose Column, row and diagonally reduced forms of a polynomial matrix

Syntax

```
[D,rk,U,Ui] = colred(A[,tol])
[D,rk,U,Ui] = colred(A,'ref'[,tol])
[D,rk,U,Ui] = colred(A,'bas'[,tol])
[D,rk,U,Ui] = rowred(A[,tol])
[D,rk,U,Ui] = rowred(A,'ref'[,tol])
[D,rk,U,Ui] = rowred(A,'bas'[,tol])
[D,U,Ui] = diagred(B[,tol])
```

Description The command

```
[D,rk,U,Ui] = colred(A)
```

brings the polynomial matrix A into column reduced form $D = AU$, where U is a unimodular matrix. The columns of D are arranged according to decreasing degrees. If the input matrix A does not have full column rank then it is reduced to the form

$$D = [D_0 \ 0]$$

with **D0** column reduced. The scalar **rk** returns the number of non-zero columns in D . **ui** is the inverse of U .

The command

[D,rk,U,Ui] = rowred(A)

brings the polynomial matrix A into row reduced form $D = UA$, where U is a unimodular matrix. The rows of D are arranged according to decreasing degrees. If the input matrix A does not have full row rank then it is reduced to the form

$$D = \begin{bmatrix} D0 \\ 0 \end{bmatrix}$$

with **D0** row reduced. The scalar **rk** returns the number of nonzero rows in D , and **ui** is the inverse of U .

Both for **colred** and **rowred** the default method '**ref**' is by repeated extraction of factors. A second method '**bas**' is based on the calculation of a minimal basis.

An optional tolerance **tol** may be specified as an additional input argument. Its default value is the global zeroing tolerance.

If Z is a continuous-time para-Hermitian polynomial matrix ($Z = Z^*$) then the command

[ZR,U,Ui] = diagred(Z)

produces a diagonally reduced para-Hermitian matrix \mathbf{ZR} and a unimodular matrix U such that

$$\mathbf{ZR} = U'ZU$$

The unimodular matrix U^{-1} is the inverse of U .

An optional tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

The polynomial matrix

```
M = [ 1  s
      0  1 ];
```

is not column reduced because its column leading coefficient matrix

```
lcoef(M,'col')
```

```
ans =
```

```
1      1
0      0
```

does not have full column rank. To make M column reduced type

```
MR = colred(M)
```

MATLAB responds with

Constant polynomial matrix: 2-by-2

MR =

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

As the next example we determine a row reduced form of the polynomial matrix

$$N = \begin{bmatrix} 1+s & s \\ 0 & 1 \\ 1+s & 0 \\ 0 & 1 \end{bmatrix}$$

The command

`NR = rowred(N)`

results in

NR =

$$\begin{bmatrix} 1 + s & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

0 0

As final example we consider the diagonal reduction of the para-Hermitian matrix

```
Z = [ 1      1+s^2
      1+s^2  1+s^4 ];
```

The command

```
[ZR,U,Ui] = diagred(Z)
```

results in the output

```
ZR =
    1      1
    1      1 - 2s^2

U =
    1      -s^2
    0      1

Ui =
    1      s^2
    0      1
```

Algorithm

The routine `colred` is dual to `rowred`.

With the option `'ref'` the routine `rowred` use the repeated factor extraction procedure described by Callier (1985). The algorithm produces both the reduction matrix U and its inverse U^{-1} .

With the option `'bas'` the row reduction algorithm is based on the computation of the minimal polynomial basis for the null space of a polynomial matrix closely related to the matrix that is to be reduced (Beelen *et al*, 1988). The algorithm produces U . Its inverse U^{-1} — if requested — is computed by application of `inv`.

The macro `diagred` uses Callier's (1985) factor extraction method.

Diagnostics

The macros `colred` and `rowred` issue error messages if

- an invalid input argument is encountered
- invalid `Not-a-Number` or `Infinite` entries are found in the input matrices
- the reduction fails
- an incorrect kernel degree is encountered

In the latter two cases the tolerance should be modified.

The macro `diagred` displays error messages if

- the input is not a “continuous-time” matrix
- the input matrix is not square

- the input matrix is singular

See also

echelon echelon form of a polynomial matrix

hermite Hermite form of a polynomial matrix

compan

Purpose Companion matrix to a polynomial matrix

Syntax

```
[C,D] = compan(A)
```

```
[C,D] = compan(A,'bot')
```

```
[C,D] = compan(A,'top')
```

```
[C,D] = compan(A,'right')
```

```
[C,D] = compan(A,'left')
```

Description Given a square polynomial matrix

$$A = A_0 + A_1*s + \dots + A_N*s^N$$

with nonsingular leading coefficient **AN**, the commands

```
[C,D] = compan(A)
```

```
[C,D] = compan(A,'bot')
```

return its “bottom” block companion matrix

$$C = \begin{bmatrix} 0 & I & 0 & \cdots & \cdots \\ 0 & 0 & I & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & I \\ -AN \setminus A0 & -AN \setminus A1 & \cdots & \cdots & -AN \setminus A(N-1) \end{bmatrix}$$

The matrix D is an identity matrix of the same dimensions as C .

Similarly, the commands

```
[C,D] = compan(A,'top')
```

```
[C,D] = compan(A,'right')
```

```
[C,D] = compan(A,'left')
```

return “top,” “right” and “left” versions of the block companion matrix, respectively.

If, in addition, the string '**sep**' is used as an input argument then the resulting block companion matrix is returned as two separate matrices, for instance

$$C = \begin{bmatrix} 0 & I & 0 & \cdots & \cdots \\ 0 & 0 & I & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & I \\ -A0 & -A1 & \cdots & \cdots & -A(N-1) \end{bmatrix}, \quad D = \begin{bmatrix} I & 0 & \cdots & \cdots & \cdots \\ 0 & I & 0 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & I & 0 \\ 0 & \cdots & \cdots & 0 & AN \end{bmatrix}$$

Here $\mathbf{A}\mathbf{N}$ may be singular.

Examples

Consider the polynomial matrix

```
A = [ 0      1
      2*s^2  3+4*s ];
```

The leading coefficient matrix

```
A{2}
ans =
      0      0
      2      0
```

is singular so we need the “separated” companion matrix

```
[C,D] = compan(A,'sep')
```

which is given by

```
C =
      0      0      1      0
      0      0      0      1
      0     -1      0      0
```

```

      0      -3      0      -4
D =
      1      0      0      0
      0      1      0      0
      0      0      0      0
      0      0      2      0

```

The polynomial matrix

```

Q = [ 1+s      s^2
      2+s^2    1 ];

```

has a nonsingular leading coefficient matrix. Its “left” block companion matrix is

```

compan(Q,'left')

ans =
      0      -1      1      0
      0      0      0      1
      0      -1      0      0
     -1      -2      0      0

```

Diagnostics

The macro issues error messages if

- an invalid syntax is used
- the input matrix is not square
- an illegal input string is encountered
- the leading coefficient matrix is singular

See also

`roots` roots of a polynomial matrix

conj

Purpose Complex conjugate of a polynomial matrix

Syntax $\mathbf{Ac} = \text{conj}(\mathbf{A})$

Description For a polynomial matrix A , the command $\mathbf{Ac} = \text{conj}(\mathbf{A})$ returns a polynomial matrix \mathbf{Ac} where all scalar coefficients are the complex conjugate of the corresponding coefficients of A .

Examples Consider

```

P = [(1+i)*s (1-i)*s]

P =

      (1+1i)s      (1-1i)s

```

The conjugate of this polynomial matrix is

```

conj(P)

ans =

      (1-1i)s      (1+1i)s

```

Algorithm If A is a polynomial matrix then $\text{conj}(A) = \text{real}(A) - i\text{imag}(A)$.

Diagnostics

The macro displays an error message if it has too many input arguments or too many output arguments.

See also

`ctranspose (')` conjugate transpose of a polynomial matrix

`transpose (. ')` transpose of a polynomial matrix

`real` real part of a polynomial matrix

`imag` imaginary part of a polynomial matrix

ctranspose (')

Purpose Complex conjugate transpose of a polynomial matrix

Syntax $\mathbf{A}^t = \mathbf{A}'$

$\mathbf{A}^t = \text{ctranspose}(\mathbf{A})$

$[\mathbf{A}^t, n] = \text{ctranspose}(\mathbf{A})$

Description For a polynomial matrix A , the commands

$\mathbf{A}^t = \mathbf{A}'$

$\mathbf{A}^t = \text{ctranspose}(\mathbf{A})$

return the a polynomial matrix \mathbf{A}^t that is formed by replacing all scalar coefficients by their complex conjugates and transposing the resulting matrix. Moreover

- if A is a polynomial matrix in s then s is replaced with $-s$
- if A is a polynomial matrix in z then z is replaced with $1/z$ and the matrix is multiplied by z^n , with n the degree of the polynomial matrix
- if A is a polynomial matrix in d, q or z^{-1} then it is similarly transformed

The command

$[\mathbf{A}^t, n] = \text{ctranspose}(\mathbf{A})$

additionally returns the degree n of A .

Examples

Consider

```
P = [(1+i)*s (1-i)*s+s^2];
```

The complex conjugate transpose of this polynomial matrix is

```
P'
```

```
ans =
```

```
-(1-1i)s
```

```
-(1+1i)s + s^2
```

If on the other hand

```
P = [(1+i)*z (1-i)*z+z^2];
```

then

```
[Pt,n] = ctranspose(P)
```

results in

```
Pt =
```

```
(1-1i)z
```

```
1+0i + (1+1i)z
```


$$n =$$

$$2$$
Algorithm

The macro `ctranspose` uses standard MATLAB operations.

Diagnostics

The macro displays an error message if it has too many input arguments or too many output arguments.

See also

<code>conj</code>	conjugate of a polynomial matrix
<code>transpose (.)</code>	transpose of a polynomial matrix
<code>real</code>	real part of a polynomial matrix
<code>imag</code>	imaginary part of a polynomial matrix

d, p, q, s, v, z, zi

Purpose Create simple basic polynomials

Syntax `P = d`

`P = p`

`P = q`

`P = s`

`P = v`

`P = z`

`P = zi`

Description The functions without argument `d`, `p`, `q`, `s` and `z` create a polynomial containing the first power of the variable that is called.

The function without argument `zi` creates the polynomial z^{-1} .

The function `v` creates a polynomial containing the first power of the current global variable.

Since `d`, `p`, `q`, `s`, `v`, `zi` and `z` are functions they may be overridden and used as variables.

Examples

Typing

```
P = z+z^3, Q = zi^4, R = 1+v^2
```

results in

```
P =
```

```
z + z^3
```

```
Q =
```

```
z^-4
```

```
R =
```

```
1 + s^2
```

Note that typing

```
D
```

returns

```
ans =
```

```
d
```

The reason is that functions are case insensitive in MATLAB so that `d` and `D` are synonymous. Also note that in spite of this

Q

is returned as

$Q =$

z^{-4}

because Q as a function has been overridden by the variable Q previously defined.

Algorithm

The macro uses standard MATLAB 5 commands.

Diagnostics

The macros return no error messages.

debe

Purpose Polynomial solution to the deadbeat regulator design

Syntax

```
[Nc,Dc] = debe(N,D)
[Nc,Dc] = debe(N,D,'r')
[Nc,Dc,E,F,degT] = debe(N,D)
[Nc,Dc,E,F,degT] = debe(N,D,'r')
```

Description Given a discrete-time plant transfer matrix

$$P(z) = D^{-1}(z)N(z)$$

the command

```
[Nc,Dc] = debe(N,D)
```

computes a deadbeat regulator with transfer matrix

$$Q(z) = N_C(z)D_C^{-1}(z) ,$$

that is, a feedback controller as in Fig. 1 such that the resulting closed-loop response to any initial condition as well as to any finite length disturbance vanishes in a finite number of steps.

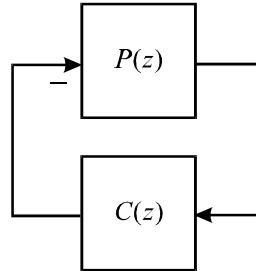


Fig. 1. Feedback structure

Similarly, for a discrete-time plant transfer matrix

$$P(z) = N(z)D^{-1}(z)$$

the command

$$[\mathbf{Nc}, \mathbf{Dc}] = \text{debe}(\mathbf{N}, \mathbf{D}, 'r')$$

computes a deadbeat regulator having transfer matrix described by

$$C(z) = D_C^{-1}(z)N_C(z) .$$

The macro works similarly for other discrete-time type operators q, d, z^{-1} .

If they exists, other deadbeat regulators may be achieved by the parameterization

$$C(z) = (N_C(z) + E(z)T(z))(D_c(z) - F(z)T(z))^{-1}$$

which is computed by the command

```
[Nc,Dc,E,F,degT] = debe(N,D),
```

or by the parameterization

$$C(z) = (D_c(z) - T(z)F(z))^{-1}(N_c(z) + T(z)E(z))$$

which is computed by the command

```
[Nc,Dc,E,F,degT] = debe(N,D,'r')
```

$T(z)$ is an arbitrary polynomial matrix parameter of compatible size with degree limited by **degT**. Any such choice of $T(z)$ results in a proper controller yielding the desired dynamics.

When the design is being made in backward shift operators (d or z^{-1}) then the degree of T is unlimited. Any choice of T results in a causal controller that guarantees a finite step response (with the number of steps depending on the degree, of course). Hence, the output argument **degT** is useless and empty in such a case. For more details on deadbeat design see Kucera and Sebek (1984), and Kucera (1979, 1991).

Notice: If the plant is strictly proper (or causal) then the resulting controller is always proper (causal). Otherwise its properness (causality) is not guaranteed and should be checked separately.

Example 1

Consider a simple third order discrete-time plant with the scalar strictly proper transfer function $P(z) = D^{-1}(z)N(z)$ given by

```
N = pol([1 1 1],2,'z')
```

```
N =
```

```
1 + z + z^2
```

```
D = pol([4 3 2 1],3,'z')
```

```
D =
```

```
4 + 3z + 2z^2 + z^3
```

A deadbeat regulator is designed by simply typing

```
[Nc,Dc] = debe(N,D)
```

```
Nc =
```

```
-2.3 - 2.3z - 2.7z^2
```

```
Dc =
```

```
0.57 + 0.71z + z^2
```

Computing the closed-loop characteristic polynomial

```
D*Dc+N*Nc
```

```
ans =
```

```
z^5
```


reveals finite modes only and hence confirms the desired deadbeat performance.

Trying

```
[Nc,Dc,E,F,degT] = debe(N,D)
```

```
Nc =
```

```
-2.3 - 2.3z - 2.7z^2
```

```
Dc =
```

```
0.57 + 0.71z + z^2
```

```
E =
```

```
4 + 3z + 2z^2 + z^3
```

```
F =
```

```
1 + z + z^2
```

```
degT =
```

```
-Inf
```

shows that (as `degT = -Inf`) there is no other proper deadbeat regulator such that the resulting system is of order 5. Higher order deadbeat controllers can be found by making the design in `d` or by solving directly the associated Diophantine equation, for instance,

```
[Dc,Nc,F,E] = axbyc(D,N,z^6)
```

```
Dc =
```

```
-0.47 + 0.1z + 0.25z^2 + z^3
```

```
Nc =
```

```
1.9 - 0.88z - 1.4z^2 - 2.2z^3
```

```
F =
```

```
-1 - z - z^2
```

```
E =
```

```
4 + 3z + 2z^2 + z^3
```

This results in a set of third order controllers parameterized by a constant T .

For a comparison, perform now the same design in the backward-shift operator z^{-1} . To convert the plant transfer function into z^{-1} , type

```
[Nneg,Dneg] = reverse(N,D);
```

```
symbol(Nneg,'z^-1');symbol(Dneg,'z^-1');Nneg,Dneg
```

```
Nneg =
```

```
z^-1 + z^-2 + z^-3
```

Dneg =

$$1 + 2z^{-1} + 3z^{-2} + 4z^{-3}$$

Typing

[Ncneg,Dcneg] = debe(Nneg,Dneg)

Ncneg =

$$-2.7 - 2.3z^{-1} - 2.3z^{-2}$$

Dcneg =

$$1 + 0.71z^{-1} + 0.57z^{-2}$$

leads to the same regulator as above (the only deadbeat regulator of second order). Higher order causal regulator can be obtained from

[Ncneg,Dcneg,Eneg,Fneg,degTneg] = debe(Nneg,Dneg)

Ncneg =

$$-2.7 - 2.3z^{-1} - 2.3z^{-2}$$

Dcneg =

$$1 + 0.71z^{-1} + 0.57z^{-2}$$

Eneg =

```

0.25 + 0.5z^-1 + 0.75z^-2 + z^-3

Fneg =

0.25z^-1 + 0.25z^-2 + 0.25z^-3

degTneg =

[]

```

A parameter $T(d)$ of any degree may be used. For instance, the choice of $T(d) = 1$ yields the third order regulator with

```

Nc_other = Ncneg+Eneg

Nc_other =

-2.5 - 1.8z^-1 - 1.5z^-2 + z^-3

Dc_other = Dcneg-Fneg

Dc_other =

1 + 0.46z^-1 + 0.32z^-2 - 0.25z^-3

```

The final check

```

Dneg*Dc_ other+Nneg*Nc_other

Constant polynomial matrix: 1-by-1

```

```
ans =
```

```
1
```

confirms the deadbeat performance.

Example 2

For the two-input two-output plant with transfer matrix $P(z) = N(z)D^{-1}(z)$ given by

```
N = [1-z z; 2-z 1]
```

```
N =
```

```
1 - z      z
```

```
2 - z      1
```

```
D = [1+2*z-z^2  -1+z+z^2; 2-z 2+3*z+2*z^2]
```

```
D =
```

```
1 + 2z - z^2    -1 + z + z^2
```

```
2 - z           2 + 3z + 2z^2
```

the deadbeat regulator is found in the form $C(z) = D_C^{-1}(z)N_C(z)$ by typing

```
[Nc,Dc] = debe(N,D,'r')
```

```
Nc =
```

```
-3.3 - 2.7z      0.59 - 1.2z
```

```

0.33 - 0.35z      0.41 + 0.22z
Dc =
1.4 - z          0.39 + 0.5z
-0.37           -0.39 + 0.5z

```

Indeed, the resulting closed-loop denominator matrix

```

Dc*D+Nc*N
ans =
z^3      0
0        z^3

```

reveals that only finite step modes are present.

Algorithm

Given $P = D^{-1}N$, the macro computes $C = N_C D_C^{-1}$ by solving the linear polynomial matrix equation $DD_C + NN_C = R$, where the right hand side matrix is either

$$R = \text{diag}\{z^i\}$$

or $R = I$ when working in forward or backward shift operators, respectively.

Diagnostics

The macro returns error messages in the following situations

- The input arguments are not polynomial objects

- The input matrices have inconsistent dimensions
- The input string is incorrect
- The system is continuous-time
- The input polynomial matrices are not coprime

See also

<code>pplace</code>	pole placement
<code>stab</code>	stabilizing controllers

deg

Purpose	Various degree matrices and leading coefficient matrices of a polynomial matrix
Syntax	<pre>[D,L] = deg(A)</pre> <pre>[D,L] = deg(A,'mat')</pre> <pre>[D,L] = deg(A,'ent')</pre> <pre>[D,L] = deg(A,'row')</pre> <pre>[D,L] = deg(A,'col')</pre> <pre>[D,L] = deg(A,'dia')</pre>
Description	<p>The commands <code>deg(A)</code> and <code>deg(A,'mat')</code> return the degree of A.</p> <p>Similarly, <code>deg(A,'ent')</code>, <code>deg(A,'row')</code> and <code>deg(A,'col')</code> return the matrix of degrees of each entry, the column vector of row degrees and the row vector of column degrees of A, respectively.</p> <p>For a para-Hermitian polynomial matrix <code>deg(A,'dia')</code> returns the vector of half diagonal degrees.</p> <p>In all the syntaxes the corresponding leading coefficient matrix is returned as the second output argument L.</p>

Examples

The simple polynomial matrix

```
P = [ 1  s
      s^2 0 ];
```

has degree and leading coefficient matrix

```
[degP,L] = deg(P)
```

```
degP =
```

```
2
```

```
L =
```

```
0      0
```

```
1      0
```

The degrees of the entries and corresponding leading coefficients are

```
[DEGP,L] = deg(P,'ent')
```

```
DEGP =
```

```
0      1
```

```
2  -Inf
```

```
L =
```

```

1      1
1      0

```

The row degrees and the row leading coefficient matrix follow as

```
[DegP,L] = deg(P,'row')
```

```
DegP =
```

```

1
2

```

```
L =
```

```

0      1
1      0

```

The column degrees and column leading coefficient matrix are

```
[DegP,L] = deg(P,'col')
```

```
DegP =
```

```

2      1

```

```
L =
```

```

0      1

```

1 0

Finally,

```
[DegZ,L] = deg([ 1+2*s^2    3*s
                -3*s       4 ],'dia')
```

results in

DegZ =

1

0

L =

-2 -3

-3 4

Note that `deg` does not zero any coefficient and tests on exact zeros. Thus,

```
deg(1+1e-32*s)
```

returns

ans =

1

Algorithm	The macro <code>deg</code> uses standard MATLAB operations.
Diagnostics	<p>The macro displays an error message if</p> <ul style="list-style-type: none"> ▪ an unknown option is encountered ▪ in case the option is '<code>dia</code>': if the input matrix is not square or not para-Hermitian
See also	<code>lcoef</code> leading coefficient matrix of a polynomial matrix

det

Purpose Polynomial matrix determinant

Syntax

```
d = det(A)
```

```
d = det(A,method)
```

```
d = det(A,tol)
```

```
d = det(A,method,tol)
```

Description The macro works similarly to the standard MATLAB function `det`. The command

```
d = det(A)
```

returns the determinant of the square polynomial matrix A . The command

```
d = det(A,method)
```

allows the user to specify the method. The following methods are available:

`'fft'` interpolation and fast Fourier transform – default

`'eig'` as the characteristic polynomial of the block companion matrix corresponding to A

The optional input parameter `tol` is the tolerance used for zeroing. Its default value is the global zeroing tolerance.

Examples

Obviously the determinant of the polynomial matrix

$$P(s) = \begin{bmatrix} 1 & s \\ s & s^2 \end{bmatrix}$$

is zero. Indeed,

```
det([1 s; s s^2])
```

returns

```
Zero polynomial matrix: 1-by-1, degree: -Inf
ans =
0
```

Algorithm

In the `'fft'` method first $P(s_j)$ is evaluated at a suitably chosen set of Fourier points s_j using the Fast Fourier Transform algorithm. Then the determinants of the resulting constant matrices are evaluated, and finally the desired determinant $\det P(s)$ is recovered from the constant determinants using the inverse FFT (Bini and Pan, 1994; Hromčík and Sebek, 1998).

In the `'eig'` algorithm the determinant is computed as the characteristic polynomial of the block companion matrix corresponding to A (Kwakernaak and Sebek, 1994).

Diagnostics

The macro displays an error message if

- an unknown option is encountered, or if
- the input matrix is not square

See also

`roots` roots of a polynomial matrix

det2d

Purpose Determinant of polynomial matrix in two variables

Syntax $q = \text{det2d}(P0, P1, \dots, Pn)$
 $q = \text{det2d}(P0, P1, \dots, Pn, tol)$

Description A polynomial matrix in two variables

$$P(v, w) = P_{00} + P_{10}v + P_{01}w + \dots + P_{mn}v^m w^n$$

is called a two-dimensional or simply 2-D polynomial matrix. If the matrix $P(v, w)$ is square then its determinant is a scalar 2-D polynomial

$$\det P(v, w) = q(v, w) = q_{00} + q_{10}v + q_{01}w + \dots + q_{kl}v^k w^l$$

defined as usual.

Alternatively, $P(v, w)$ may be described as a polynomial matrix in w having as its coefficients polynomial matrices in v denoted by $P_i(v)$

$$P(v, w) = P(v)(w) = P_0(v) + P_1(v)w + P_2(v)w^2 + \dots + P_n(v)w^n$$

and also its determinant may be written as

$$\det P(v, w) = q(v, w) = q_0(v) + q_1(v)w + q_2(v)w^2 + \dots + q_l(v)w^l.$$

For such a $P(v, w)$ given by $P_0(v)$, $P_1(v)$, \dots , $P_n(v)$ the command

q = det2d(P0,P1,...,Pn)

returns a vector q of polynomial coefficients $q_0(v)$, $q_1(v)$, \dots , $q_l(v)$ of the determinant $q(v, w) = \det P(v, w)$.

Note: The macro works with v equal to any of the standard symbols s, p, z, q, z^{-1}, d . The other symbol w is typically an uncertain real parameter encountered in robust stability analysis for single-parameter uncertain polynomial matrices.

The macro may also be called as

q = det2d(P0,P1,...,Pn,tol)

where the optional input parameter **tol** is the tolerance used for zeroing. Its default value is the global zeroing tolerance.

Note: If the last input argument is a scalar then it is always considered to be the tolerance.

Example

Obviously the determinant of the polynomial matrix

$$P(s) = \begin{bmatrix} 1+r & s \\ sr^2 & s^2r \end{bmatrix} = \begin{bmatrix} 1 & s \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & s^2 \end{bmatrix}r + \begin{bmatrix} 0 & 0 \\ s & 0 \end{bmatrix}r^2$$

is

$$\det P(s, r) = q(s, r) = s^2r$$

and we should get $q_0(s)=0$, $q_1(s)=s^2$ and $q_i(s)=0$ for all higher powers of r .
Indeed,

```
q = det2d([1 s;0 0],[1 0;0 s^2;],[0 0;s 0])
```

returns

```
q =  
  
0  
  
s^2
```

as expected.

Algorithm

The macro is based on an original application of the 2-D Fast Fourier Transform (FFT) routine.

Diagnostics

The macro displays an error message if the input matrices are not square or of the same size. It displays a warning message if the input matrices are not in the same variable symbol.

See also

`det` polynomial matrix determinant

diag

Purpose Extract polynomial matrix diagonals or create diagonal polynomial matrices

Syntax

```
a = diag(A)
a = diag(A,k)
A = diag(a)
A = diag(a,k)
```

Description If A is a polynomial matrix with at least two columns and two rows then

```
a = diag(A)
```

returns a column vector whose entries are the diagonal entries of A . If the command is

```
a = diag(A,k)
```

with $k \geq 0$, then the k th diagonal above the main diagonal is returned, and if $k < 0$ then the $|k|$ th diagonal below the main diagonal is returned.

If a is a column or row vector of polynomials of length n then

```
A = diag(a)
```

returns an $n \times n$ polynomial matrix A with the entries of a on its main diagonal. The command

```
A = diag(a,k)
```

returns a square matrix A that has the entries of a on its k th diagonal, where again k may be positive or negative.

Examples

The command

```
diag([ 1+s    2*s^2
      -2     3+4*s^3 ])
```

produces

```
ans =
      1 + s
      3 + 4s^3
```

while

```
diag([ 1+s    2+4*s^3 ],1)
```

results in

```
ans =
      0      1 + s      0
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 2 + 4s^3 & 0 \\ 0 & 0 \end{bmatrix}$$

Note that if X is a polynomial matrix whose size is at least 2×2 then `diag(diag(X))` is a diagonal matrix whose diagonal is identical to that of X , and `sum(diag(X))` is the trace of X .

Algorithm

The routine uses standard routines from the Polynomial Toolbox.

Diagnostics

The macro displays no error messages.

See also

`tril` extract the lower triangular part

`triu` extract the upper triangular part

display

Purpose Command window display of a polynomial matrix

Syntax `display(P)`

Description The command

`display(P)`

displays the polynomial matrix P in the MATLAB command window. The display format is set by the macro `pformat`.

Examples We display the matrix

```
P = [1+2*s 4/3*s^2];
```

in different formats:

```
pformat coef, display(P)
```

```
Polynomial matrix in s: 1-by-2, degree: 2
```

```
P =
```

```
Matrix coefficient at s^0 :
```

```
1      0
```

Matrix coefficient at s^1 :

2 0

Matrix coefficient at s^2 :

0 1.3333

`pformat block, display(P)`

Polynomial matrix in s : 1-by-2, degree: 2

$P =$

1.0000 0 2.0000 0 0
1.3333

`pformat symb, display(P)`

$P =$

1 + 2s 1.3333s²

`pformat symbr, display(P)`

$P =$

1 + 2*s 4/3*s²

Algorithm

The macro uses standard MATLAB 5 operations.

Diagnostics

The macro displays no error messages

See also

`char` convert a polynomial matrix to strings

`pformat` switch between different display formats

dsp2pol

Purpose Conversion of a polynomial or polynomial matrix in DSP format to Polynomial Toolbox format

Syntax `P = dsp2pol(M)`

Description The command

`P = dsp2pol(M)`

converts the polynomial or polynomial matrix M in DSP format to Polynomial Toolbox format.

In DSP format a polynomial is represented as a row vector whose entries are the coefficients of the polynomial according to ascending powers. A polynomial matrix is a cell array of the same dimensions as the polynomial matrix, with each cell a row vector representing the corresponding entry of the polynomial matrix in DSP format. The DSP format is typically used for discrete-time systems.

Examples *Conversion of a polynomial from DSP format to Polynomial Toolbox format:*

```
gprop q; P = dsp2pol([1 2 3])
```

```
P =
```

```
1 + 2q + 3q^2
```

Conversion of a polynomial matrix from DSP format:

M is a matrix in DSP format. First display M and its entries:

```
M
M =
      [1x3 double]      [1x3 double]
M{1,1}, M{1,2}
ans =
      1      2      0
ans =
      3      0      4
```

Next convert to Polynomial Toolbox format:

```
P = dsp2pol(M)
P =
      1 + 2q      3 + 4q^2
```

Algorithm

The macro `dsp2pol` uses standard MATLAB 5 operations.

Diagnostics

The macro `dsp2pol` displays an error message if the input is not a polynomial or polynomial matrix in DSP format.

See also

<code>mat2pol</code>	conversion from Polynomial Toolbox format to MATLAB or Control System Toolbox format
<code>pol2dsp</code>	conversion from Polynomial Toolbox format to DSP format
<code>pol2mat</code>	conversion from to MATLAB or Control System Toolbox format to Polynomial Toolbox format

dss2lmf, dss2rmf

Purpose Conversion of descriptor representation to left or right matrix fraction representation

Syntax

```
[N,D] = dss2lmf(a,b,c,d,e)
[N,D] = dss2lmf(a,b,c,d,e,tol)
[N,D] = dss2rmf(a,b,c,d,e)
[N,D] = dss2rmf(a,b,c,d,e,tol)
```

Description The commands

```
[N,D] = dss2lmf(a,b,c,d,e)
[N,D] = dss2lmf(a,b,c,d,e,tol)
```

convert the transfer matrix

$$H(s) = c(se - a)^{-1}b + d$$

of the descriptor system

$$\begin{aligned} e\dot{x} &= ax + bu \\ y &= cx + du \end{aligned}$$

to the left coprime polynomial matrix fraction

$$H(s) = D^{-1}(s)N(s)$$

If H is proper then the denominator matrix D is row reduced .

The commands

```
[N,D] = dss2lmf(a,b,c,d,e)
```

```
[N,D] = dss2lmf(a,b,c,d,e,tol)
```

produce the right polynomial matrix fraction

$$H(s) = N(s)D^{-1}(s)$$

If H is proper then D is column reduced.

The optional input parameter `tol` is a tolerance. Its default value is the global zeroing tolerance.

The continuous-time interpretation of the input applies if the default indeterminate variable is either s or p . The variables of the output arguments N and D are set accordingly.

In a discrete-time environment, where either z or p is the default indeterminate variable, the input parameters define the discrete-time descriptor system

$$\begin{aligned} ex(t+1) &= ax(t) + bu(t) \\ y(t) &= cx(t) + u(t) \end{aligned}$$

The output parameters N and D then are polynomial matrices in z or q representing the transfer matrix in the left or right fractional form $H = D^{-1}N$ or $H = ND^{-1}$.

If the default indeterminate variable is z^{-1} or d then the input parameters again are taken to define a discrete-time descriptor system, and the output parameters N and D are converted to polynomial matrices in the default variable.

Examples

Consider the SISO descriptor system defined by

```
a = [ 1      0      0
      0      1      0
      0      0      1 ];

b = [ 0
      0
      1 ];

c = [-1      0      0 ];

d =  0;

e = [ 0      1      0
      0      0      1
      0      0      0 ];
```

The system may be converted to matrix fraction form by the command

```
[N,D] = dss2lmf(a,b,c,d,e)
```

```
N =
```

```
    s^2
```

```
Constant polynomial matrix: 1-by-1
```

```
D =
```

```
    1
```

If we redefine the default indeterminate variable according to

```
gprop 'z^-1'
```

then the command

```
[N,D] = dss2lmf(a,b,c,d,e)
```

results in

```
Constant polynomial matrix: 1-by-1
```

```
N =
```

```
    1
```

```
D =
```

$$z^{-2}$$

Note that this system is not causal.

Algorithm

Given the transfer matrix

$$H(s) = c(se - a)^{-1}b + d$$

the routine **lmf2rmf** is used for the conversion

$$(se - a)^{-1}b = N_r(s)D_r^{-1}(s)$$

The desired left matrix fraction then follows by application of **rmf2lmf** to

$$H(s) = (cN_r(s) + d(s)D_r(s))D_r(s) = D^{-1}(s)N(s)$$

The second conversion is necessary to make sure that the fraction is coprime.

This algorithm also applies if the indeterminate variable is p , z or q . If the indeterminate variable is z^{-1} or d then a final conversion of the fraction with the help of **reverse** is needed.

The **dss2rmf** conversion follows similarly.

Diagnostics

Error messages are issued under the following conditions

- the number of input arguments is incorrect
- the input matrices have incompatible sizes

- the input matrices are not constant matrices
- the tolerance is invalid
- the matrix $se-a$ is singular

See also

lmf2dss, rmf2dss	conversion of to a left or right polynomial matrix fraction to descriptor representation
ss2rmf, rmf2ss	conversion from state space representation to right or
ss2lmf, lmf2ss	left matrix fraction or vice-versa

dss2ss

Purpose Conversion of descriptor to (generalized) state space representation

Syntax `[a,b,c,d] = dss2ss(A,B,C,D,E)`
`[a,b,c,d] = dss2ss(A,B,C,D,E,tol)`

Description The commands

`[a,b,c,d] = dss2ss(A,B,C,D,E)`
`[a,b,c,d] = dss2ss(A,B,C,D,E,tol)`

convert the descriptor system

$$E\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

to the state space system

$$\dot{x} = ax + bu$$

$$y = cx + d(s)u$$

with $d(s)$ a polynomial matrix and s the differentiation operator. The systems are equivalent in the sense that

$$C(sE - A)^{-1}B + D = c(sI - a)^{-1}b + d(s)$$

If instead of s the default indeterminate variable is p then the system is also a continuous-time system but d is returned as polynomial in p .

If the default indeterminate variable is z , q , z^{-1} or d then the input parameters are considered to represent the discrete-time descriptor system

$$\begin{aligned}Ex(t+1) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t)\end{aligned}$$

The output parameters then define the generalized discrete-time state space system

$$\begin{aligned}\mathbf{x}(t+1) &= \mathbf{a}\mathbf{x}(t) + b\mathbf{u}(t) \\ y(t) &= c\mathbf{x}(t) + d(z)\mathbf{u}(t)\end{aligned}$$

with z the time-shift operator defined by $zu(t) = u(t+1)$. If the default indeterminate variable is q then $d()$ is returned as a polynomial in q , if the default indeterminate variable is z^{-1} then $d()$ is returned as a polynomial matrix in z , and if the default indeterminate is d then $d()$ is returned as a polynomial matrix in q .

If the output parameter d is constant or the zero matrix then it is returned as an ordinary matrix in MATLAB format.

The input parameter `tol` is an optional tolerance parameter, which is used to separate the finite from the infinite poles of the system. Its default value is the global zeroing tolerance.

Examples

Consider the descriptor system defined by

```

E = [ 1      0      0      0
      0      1      0      0
      0      0      1      0
      0      0      0      0 ];

A = [ 0      1      0      0
      0      0      0      0
      0      0      0      1
      0      0      1      0 ];

B = [ sqrt(2)  0
      1         1
      0         0
      0        -1 ];

C = [ 1      0      0      0
      0      0      0.1  0.01
      -1     0      0      0 ];

```

```
D = [ 1      0
      0      0
      -1     0  ];
```

Application of

```
[a,b,c,d] = dss2ss(A,B,C,D,E)
```

produces the state space system given by

a =

```
0      1
0      0
```

b =

```
-1.4142      0
-1.0000     -1.0000
```

c =

```
-1      0
0      0
1      0
```

```

d =
      1      0
      0      0.1 + 0.01s
     -1      0

```

As a second example consider the descriptor system given by

```

E = [  1  0  0
      0  1  0
      0  0  0 ];

A = [  0  1  0
      0  0  1
     -1  0  0 ];

B = [  0
      0
      1 ];

C = [  0  0  1 ];

D = 0;

```

The command

```
[a,b,c,d] = dss2ss(A,B,C,D,E)
```

results in

```
a =  
  
[]  
  
b =  
  
Empty matrix: 0-by-1  
  
c =  
  
Empty matrix: 1-by-0  
  
d =  
  
s^2
```

Evidently the descriptor system defines a system with the purely polynomial transfer function

$$H(s) = s^2$$

The command

```
gprop 'z^-1', [a,b,c,d] = dss2ss(A,B,C,D,E)
```

returns

```
a =
    []

b =
    Empty matrix: 0-by-1

c =
    Empty matrix: 1-by-0

d =
    z^2
```

Algorithm

Using the routine `pencan` the pencil $sE-A$ is transformed to the Kronecker canonical form

$$q(sE-A)z = \begin{bmatrix} sI-a & 0 \\ 0 & I-se \end{bmatrix}$$

with e nilpotent. Partitioning

$$Cz = [c \quad c_o], \quad qB = \begin{bmatrix} b \\ b_o \end{bmatrix}$$

it follows that

$$C(sE - A)^{-1}B + D = c(sI - a)^{-1}b + c_o(I - se)^{-1}b_o + D$$

Because e is nilpotent we have

$$(I - se)^{-1} = I + se + se^2 + \dots + s^p e^p$$

for some finite power p which easily can be determined. This immediately leads to the desired result.

Diagnostics

The routine displays error messages under the following circumstances:

- An invalid value for the tolerance is encountered
- The number of input arguments incorrect
- The input matrices are not constant matrices
- The input matrices have incompatible sizes

See also

ss2dss	conversion of (generalized) state space representation to descriptor representation
dss	create descriptor state space models or convert LTI models to descriptor state space form

dsshinf

Purpose Computation of H-infinity suboptimal compensators for a descriptor standard plant

Syntax

```
[Ak,Bk,Ck,Dk,Ek,clpoles,rho] = ...
    dsshinf(A,B,C,D,E,nmeas,ncon,gamma)
[Ak,Bk,Ck,Dk,Ek,clpoles,rho] = ...
    dsshinf(A,B,C,D,E,nmeas,ncon,gamma[,options][,tol])
```

Description The command

```
[Ak,Bk,Ck,Dk,Ek,clpoles,rho] = ...
    dsshinf(A,B,C,D,E,nmeas,ncon,gamma)
```

computes the suboptimal compensator for the standard H_∞ problem for the generalized plant

$$E\dot{x} = Ax + B \begin{bmatrix} w \\ u \end{bmatrix}$$

$$\begin{bmatrix} z \\ y \end{bmatrix} = Cx + D \begin{bmatrix} w \\ u \end{bmatrix}$$

with **nmeas** measured outputs y and **ncon** control inputs u according to the level **gamma**.

Several options may be specified. Without the option '**a11**' the routine computes the “central” compensator

$$K(s) = Ck(sEk - Ak)^{-1}Bk + Dk$$

If the option '**a11**' is included then after partitioning

$$Bk = [Bk1 \ Bk2]$$

$$Ck = [Ck1$$

$$Ck2]$$

$$Dk = [0 \quad 0$$

$$0 \quad Dk22]$$

where **Bk1** has **nmeas** columns and **Ck1** has **ncon** rows, all suboptimal compensators may be parametrized as

$$\dot{x}_k = Ak x_k + Bk1 y + Bk2 p$$

$$u = Ck1 x_k + p$$

$$q = Ck2 x_k + y + Dk2 p$$

$$p = Uq$$

U is any stable LTI system whose transfer matrix has infinity-norm less than or equal to 1. The central compensator follows by setting $U = 0$.

The array **clpoles** contains the finite closed-loop poles obtained for the central compensator, that is, the closed poles with magnitude less than $1/\text{tol}$. The default value of the optional input parameter **tol** is $1\text{e-}6$.

The output parameter **rho** is the smallest singular value of a matrix z that arises in the algorithm. For so-called type 2 optimal solutions this matrix z is singular and, hence, **rho** = 0. This parameter may be used in searching for the optimal solution (see **dssrch**).

If **gamma** is too small so that no suboptimal compensator exists then the routine exits with all output arguments empty. In verbose mode a warning message is issued.

Conditions on the input data: If D is partitioned as

$$D = \begin{bmatrix} D11 & D12 \\ D21 & D22 \end{bmatrix}$$

where **D12** has **ncon** columns and **D21** has **nmeas** rows, then **D12** needs to have full column rank and **D21** full row rank. Use the command **dssreg** to “regularize” the system if this condition is not met.

Options: The available options (none or several may be included) besides '**all**' are

'notest' the rank tests on **D12** and **D21** are omitted

'nopoles' the closed loop poles are not computed. In this case the output argument **clpoles** is empty.

Example

We consider the mixed sensitivity problem for the SISO plant with transfer function

$$P(s) = \frac{1}{s^2}$$

and the shaping and weighting functions

$$V(s) = \frac{s^2 + s\sqrt{2} + 1}{s^2}, \quad W_1(s) = 1, \quad W_2(s) = c(1 + rs)$$

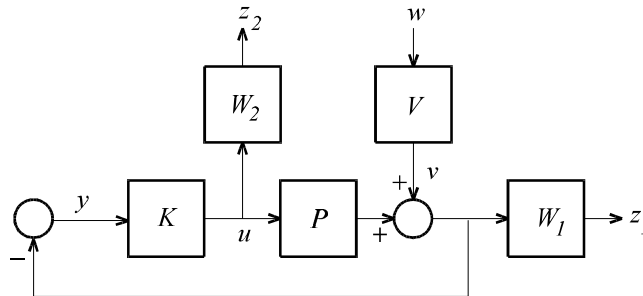


Fig. 2. The mixed sensitivity problem

(Kwakernaak, 1993). The block diagram of Fig. 2 shows the standard plant that defines this mixed sensitivity problem.

We construct a descriptor representation of the plant as follows. First we note that

$$-y = \begin{bmatrix} P & V \end{bmatrix} \begin{bmatrix} u \\ w \end{bmatrix}$$

A state representation of

$$\begin{bmatrix} P & V \end{bmatrix} = \frac{1}{s^2} \begin{bmatrix} 1 & s^2 + s\sqrt{2} + 1 \end{bmatrix}$$

may for instance be found with the help of the Polynomial Toolbox routine `lmf2ss`. We have

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & \sqrt{2} \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u \\ w \end{bmatrix}$$

$$-y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ w \end{bmatrix}$$

A descriptor representation of the differentiating filter W_2 is found by defining two pseudo state variables

$$x_3 = u$$

$$x_4 = \dot{u}$$

From this we obtain the descriptor equations

$$\begin{aligned}\dot{x}_3 &= x_4 \\ 0 &= -x_3 + u\end{aligned}$$

The corresponding output equation follows as

$$z_2 = c(1+rs)u = crx_4 + cu$$

By inspection of the block diagram we can now write the descriptor representation of the standard plant as

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_E \underbrace{\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_A \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}} + \underbrace{\begin{bmatrix} \sqrt{2} & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}}_B \underbrace{\begin{bmatrix} w \\ u \end{bmatrix}}$$

$$\underbrace{\begin{bmatrix} z_1 \\ z_2 \\ y \end{bmatrix}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & cr \\ -1 & 0 & 0 & 0 \end{bmatrix}}_C \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}} + \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & c \\ -1 & 0 \end{bmatrix}}_D \underbrace{\begin{bmatrix} w \\ u \end{bmatrix}}$$

Both $\mathbf{D12}$ and $\mathbf{D21}$ have full rank as required.

We first define the input data A , B , C , D , and E and set `nmeas` = 1, `ncon` = 1.

```

c = 0.1; r = 0.1;

E = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 0];

A = [0 1 0 0; 0 0 0 0; 0 0 0 1; 0 0 -1 0];

B = [sqrt(2) 0; 1 1; 0 0; 0 1];

C = [1 0 0 0; 0 0 0 c*r; -1 0 0 0];

D = [1 0; 0 c; -1 0];

nmeas = 1; ncon = 1; gamma = 10;

```

We next run the macro `dsshinf`:

```

[Ak,Bk,Ck,Dk,Ek,clpoles,rho] = ...
    dsshinf(A,B,C,D,E,nmeas,ncon,gamma)

```

`Ak =`

```

-0.0000    0.0000    0.0000    10.1103
-0.0733   -0.4206   -0.9820   -6.6580
-0.0660   -1.1965    1.0953   -8.3990
 0.9950   -0.0929   -0.0341    0.0099

```

`Bk =`


```

0
-0.3974
1.6859
0
Ck =
    0.0990    0.9525    0.2528    9.9731
Dk =
    0
Ek =
    0         0         0         0
    0.0073   -0.2744    0.8884   -0.2433
    0.0067    0.0256    0.3661    0.8261
   -0.0990   -0.9525   -0.2528    0.1373
clpoles =
-10.0526 + 0.0000i
-2.1935 - 2.3064i

```

```
-2.1935 + 2.3064i
```

```
-0.7071 - 0.7071i
```

```
-0.7071 + 0.7071i
```

```
rho =
```

```
0.8611
```

The descriptor representation of the suboptimal compensator is not minimal. We use the routine `dssmin` to reduce the dimension of the representation:

```
[ak,bk,ck,dk,ek] = dssmin(Ak,Bk,Ck,Dk,Ek)
```

```
ak =
```

```
-10.3436    74.0344 -178.6303
```

```
0    -7.0852    17.4110
```

```
0    -1.7125    1.5749
```

```
bk =
```

```
197.3278
```

```
-19.2045
```

```
-2.3324
```

```

ck =
    -0.0962    -0.9854    -0.0272

dk =
     0

ek =
     1     0     0
     0     1     0
     0     0     1

```

Since \mathbf{e}_k is a unit matrix this is a state representation. Because $\mathbf{d}_k = 0$ the compensator is strictly proper. This is a consequence of choosing the weighting function W_2 nonproper.

We rerun the command with the option `'all'`:

```

[Ak,Bk,Ck,Dk,Ek,clpoles,rho] = ...
    dsshinf(A,B,C,D,E,nmeas,ncon,gamma,'all');

Bk, Ck, Dk

Bk =
     0    1.0000

```

```

-0.3974    -0.6623
 1.6859    -0.7493
          0          0

Ck =

 0.0990    0.9525    0.2528    9.9731
 0.0010   -0.2262    0.4290   -0.7448

Dk =

 0      0
 0      0

```

Ek, **Ak**, **clpoles** and **rho** do not change as compared to the previous solution.

Algorithm

The solution of the H_∞ problem relies on the frequency domain solution of the standard H_∞ problem as described in Kwakernaak (1998) with a much-improved algorithm for the spectral factorization that this solution involves. The algorithm eliminates any singularities that occur at type 2 optimal solutions.

Diagnostics

The macro displays error messages in the following situations:

- The input matrices have inconsistent dimensions
- An invalid option is encountered

- A wrong last input parameter is encountered
- Too many options are specified
- **D12** is not tall or **D21** is not wide
- **D12** or **D21** does not have full rank

In verbose mode a warning is issued if **gamma** is too small so that no solution exists.

See also	dssrch	search H_∞ -optimal compensators for descriptor systems
	dssreg	regularize descriptor systems
	dssmin	minimize descriptor systems

dssmin

Purpose Minimization of a descriptor representation

Syntax `[a,b,c,d,e] = dssmin(A,B,C,D,E)`
`[a,b,c,d,e] = dssmin(A,B,C,D,E,tol)`

Description The commands

`[a,b,c,d,e] = dssmin(A,B,C,D,E)`
`[a,b,c,d,e] = dssmin(A,B,C,D,E,tol)`

serve to minimize the dimension of the pseudo state of the descriptor system

$$\begin{array}{lcl} \dot{Ex}(t) = Ax(t) + Bu(t) & & Ex(t+1) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) & \text{or} & y(t) = Cx(t) + Du(t) \end{array}$$

If the system has a proper transfer matrix then it is returned in state space form, that is, $e = I$.

The optional parameter `tol` specifies the tolerance that is used to separate the finite from the infinite poles. The default value is `1e-8`.

Note: Any finite noncontrollable or nonobservable modes of the system are preserved.

Examples

Consider the descriptor system

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \dot{x} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} u$$

$$y = [1 \ 0 \ 0]x$$

It is defined by typing

```
E = [ 1 0 0
      0 1 0
      0 0 0 ];

A = [ 0 0 0
      0 0 1
      0 -1 0 ];

B = [1; 0; 1];

C = [ 1 0 0 ];

D = 0;
```

The minimal representation follows by typing

```
[a,b,c,d,e] = dssmin(A,B,C,D,E)
```

```
a =
```

```
0
```

```
b =
```

```
-1
```

```
c =
```

```
-1
```

```
d =
```

```
0
```

```
e =
```

```
1
```

The minimal system turns out to be a state space system of dimension 1.

Algorithm

Using the routine `dss2ss` the descriptor system is converted to state space form

$$\dot{\mathbf{x}} = \mathbf{ax} + \mathbf{bu}$$

$$\mathbf{y} = \mathbf{cx} + D_o(s)\mathbf{u}$$

with $D_o(s)$ a polynomial matrix. Subsequently, **ss2dss** is invoked to determine a minimal descriptor representation.

Diagnostics

The macro **dssmin** displays error messages if the number of input arguments is incorrect or an incorrect value for the tolerance is encountered.

See also

- dss2ss** conversion from descriptor to (generalized) state space representation
- ss2dss** conversion from (generalized) state space to descriptor representation

dssrch

Purpose Search for H_∞ optimal compensators for a descriptor standard plant

Syntax

```
[Ak,Bk,Ck,Dk,Ek,gopt,clpoles] = ...
    dssrch(A,B,C,D,E,nmeas,ncon,gmin,gmax)
[Ak,Bk,Ck,Dk,Ek,gopt,clpoles] = ...
    dssrch(A,B,C,D,E,nmeas,ncon,gmin,gmax[,pars][,'show'])
```

Description The command

```
[Ak,Bk,Ck,Dk,Ek,gopt,clpoles] = ...
    dssrch(A,B,C,D,E,nmeas,ncon,gmin,gmax)
```

computes the central H_∞ optimal solution for the standard plant with the descriptor representation

$$E\dot{x} = Ax + B \begin{bmatrix} w \\ u \end{bmatrix}$$

$$\begin{bmatrix} z \\ y \end{bmatrix} = Cx + D \begin{bmatrix} w \\ u \end{bmatrix}$$

The plant has **ncon** control inputs u and **nmeas** measured outputs y . The optimal compensator has the transfer matrix representation

$$K(s) = C_k (sE_k - A_k)^{-1} B_k + D_k$$

The output parameter **gopt** is the minimal H_∞ norm and **clpoles** represents the closed-loop poles. The input parameter **gmin** is the lower limit of the initial search interval and **gmax** the upper limit.

Features

1. If no solution exists within the initial search interval [**gmin**,**gmax**] then the interval is automatically expanded.
2. The macro also computes optimal solutions for generalized plants that have unstable fixed poles.
3. The macro computes both type 1 optimal solutions (**gopt** coincides with the lowest value of **gamma** for which spectral factorization is possible) and type 2 optimal solutions (**gopt** is greater than this lowest value).
4. After the search has been completed the compensator is reduced to minimal dimension. If the compensator is proper then **E_k** is the unit matrix.
5. Initially the search is binary. If the solution is of type 2 and close to optimal then a secant method is used to obtain fast convergence to an accurate solution.

6. Before the search is initiated the descriptor representation of the generalized plant is regularized in the sense that it is modified so that **D12** and **D21** have full rank.

The optional input parameter **pars** is a vector with components

pars = [**acc** **tolrho** **thrrho** **maxpole** **tolrnk** **tolstable**]

with

acc Accuracy. If **gmax-gmin** < **acc** then the search is terminated. The default value is **1e-4**.

tolrho Tolerance on the (nonnegative) singularity parameter **rho**. A type 2 optimal solution is achieved exactly if **rho** = 0. If **rho** < **tolrho** then the search is terminated. The default value of **tolrho** is **1e-8**.

thrrho Threshold on rho. If **rho** < **thrrho** then the search switches from binary to secant. The default value of **thrrho** is 0.01.

maxpole Largest size of an open- or closed-loop pole. If the size of a pole is greater than **maxpole** then the pole is considered to be a pole at infinity. A fixed open-loop pole is considered unstable if its real part is greater than -1/**maxpole**. The default value of **maxpole** is **1e+6**.

tolrnk Tolerance in the rank test to determine whether an open-loop pole is uncontrollable or observable and, hence, a fixed pole. The default value is **1e-8**.

tolstable Tolerance used in **isstable** to test stability. The default value is **1e-8**.

If **pars** has p entries, with $p < 6$, then these entries are assigned to the first p parameters in the list. The remaining parameters take the default values.

If the optional input parameter '**show**' is present then the progress of the search is displayed.

Examples

Example 1: Double integrator plant (Kwakernaak, 1993).

We consider the mixed sensitivity problem for the SISO plant with transfer function

$$P(s) = \frac{1}{s^2}$$

and the shaping and weighting functions

$$V(s) = \frac{s^2 + s\sqrt{2} + 1}{s^2}, \quad W_1(s) = 1, \quad W_2(s) = c(1 + rs)$$

In the Example section of the description of the command **deshinf** it is shown that this problem corresponds to the standard plant given by

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_E \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \underbrace{\begin{bmatrix} \sqrt{2} & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}}_B \begin{bmatrix} w \\ u \end{bmatrix}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & cr \\ -1 & 0 & 0 & 0 \end{bmatrix}}_C \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & c \\ -1 & 0 \end{bmatrix}}_D \begin{bmatrix} w \\ u \end{bmatrix}$$

We first define the plant.

```
c = 0.1; r = 0.1;

E = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 0];

A = [0 1 0 0; 0 0 0 0; 0 0 0 1; 0 0 -1 0];

B = [sqrt(2) 0; 1 1; 0 0; 0 1];

C = [1 0 0 0; 0 0 0 c*r; -1 0 0 0];

D = [1 0; 0 c; -1 0];

nmeas = 1; ncon = 1;
```

Next, the search is started.

```
gmin = 1; gmax = 10;  
[Ak,Bk,Ck,Dk,Ek,gopt,clpoles] = ...  
    dssrch(A,B,C,D,E,nmeas,ncon,gmin,gmax,'show')
```

MATLAB responds with

```
Search for optimal solution  
  
action gamma  
-----  
  
testgmax 10  
  
testgmin 1  
  
binsearch 5.5  
  
binsearch 3.25  
  
binsearch 2.125  
  
binsearch 1.5625  
  
binsearch 1.28125  
  
binsearch 1.42188  
  
binsearch 1.35156
```

```
binsearch 1.38672
```

```
secsearch 1.38343
```

```
secsearch 1.38332
```

```
secsearch 1.38331
```

```
secsearch 1.38331
```

$A_k =$

```
-13.6965    77.7084
```

```
-0.5976    -2.3738
```

$B_k =$

```
1.0e+003 *
```

```
-1.4810
```

```
-0.0518
```

$C_k =$

```
-0.0303    -0.9789
```

$D_k =$

```
-3.2317e-007
```



```

Ek =

    1    0
    0    1

gopt =

    1.3833

clpoles =

    -7.3281 + 1.8765i
    -7.3281 - 1.8765i
    -0.7071 + 0.7071i
    -0.7071 - 0.7071i

```

In the search process, first the upper and lower bounds of the initial search interval are tested. Next, eight binary search steps follow, after which in four secant search steps the search fast and accurately reaches the optimal solution.

Because $E_k = I$ the optimal compensator is proper and because $D_k = 0$ it is even strictly proper. We compute the compensator in transfer function form as

```
[y,x] = ss2lmf(Ak,Bk,Ck)
```

```
y =
```

$$-57 - 96s$$

$$x =$$

$$-79 - 16s - s^2$$

Example 2. “Model matching” problem (Kwakernaak, 1996).

Given an unstable SISO system with transfer function P we consider the problem of finding a stable system with transfer function $K(s)$ such that

$$\|P - K\|_{\infty}$$

is minimal. It is easy to check that this corresponds to an H_{∞} -optimization problem with generalized plant

$$G = \begin{bmatrix} P & -1 \\ 1 & 0 \end{bmatrix}$$

If for instance

$$P(s) = \frac{1}{1-s}$$

then in state space form the generalized plant is given by

$$\dot{x} = x + w$$

$$z = -x - u$$

$$y = w$$

We thus define the input data according to

```
E = 1; A = 1; B = [1 0]; C = [-1; 0]; D = [0 -1; 1 0];
nmeas = 1; ncon = 1;
```

The solution is obtained as follows.

```
gmin = 1; gmax = 10;
[Ak,Bk,Ck,Dk,Ek,gopt,clpoles] = ...
    dssrch(A,B,C,D,E,nmeas,ncon,gmin,gmax,'show')
```

MATLAB responds with

```
Fixed poles
    1 unstable

Search for optimal solution

action gamma
-----
testgmax 10
testgmin 1
testgmin 0.5
```

```
testgmin 0.25  
binsearch 5.125  
binsearch 2.6875  
binsearch 1.46875  
binsearch 0.859375  
binsearch 0.554688  
binsearch 0.402344  
binsearch 0.478516  
binsearch 0.516602  
binsearch 0.497559  
secsearch 0.500027  
secsearch 0.5  
secsearch 0.5
```

Ak =

[]

Bk =

```

Empty matrix: 0-by-1

Ck =

Empty matrix: 1-by-0

Dk =

    0.5000

Ek =

    []

gopt =

    0.5000

clpoles =

    1

```

The generalized plant has a fixed unstable pole at 1. The routine first adjusts the lower limit of the search interval. After 9 binary search steps 3 secant steps are enough to reach an accurate solution. The optimal solution is given by

$$K(s) = 0.5000$$

Algorithm

Before the search is started the plant is “regularized” using the routine **desreg**. Next the upper and lower limits of the search interval are tested using **deshinf**. If

no solution exists inside this interval then the upper limit is doubled or the lower limit halved until a solution is contained in the interval.

Once the search interval is established a binary search is started, where the next solution is sought at $(g_{\max} + g_{\min})/2$ and the interval is adjusted depending on whether or not a solution exists at this point.

As soon as the output parameter `rho` of `deshinf` is less than `thrrho` the search switches to a secant search where the next solution is sought at $(g_{\max} \cdot r_{\min} + g_{\min} \cdot r_{\max}) / (r_{\max} + r_{\min})$. The search stops if either $g_{\max} - g_{\min} < acc$ or $rho < tol_{rho}$.

After the search has been completed the dimension of the compensator that has been computer is reduced using `desmin`.

Diagnostics

The macro displays error messages in the following situations:

- An invalid option is encountered
- The input parameter `par` has too many entries
- The input matrices have inconsistent dimensions
- Too many options are specified
- `gmax < gmin`
- `gmax` has been doubled 4 times without finding a stabilizing compensator at `gmax`

- `gmin` has been halved 4 times but a stabilizing compensator still exists at `gmin`

If there are any fixed poles then these are listed.

If the option '`show`' is present then it is reported if the plant is transformed in order to regularize it, and the progress of the search is displayed.

See also

<code>dsshinf</code>	H_∞ optimization for a descriptor plant
<code>dssreg</code>	“regularization” of a standard descriptor plant
<code>dssmin</code>	dimension reduction of a descriptor system

dssreg

Purpose “Regularization” of a standard descriptor plant

Syntax

$$[a,b,c,d,e] = \text{dssreg}(A,B,C,D,E,nmeas,ncon)$$
$$[a,b,c,d,e] = \text{dssreg}(A,B,C,D,E,nmeas,ncon,tol)$$

Description The commands

$$[a,b,c,d,e] = \text{dssreg}(A,B,C,D,E,nmeas,ncon)$$
$$[a,b,c,d,e] = \text{dssreg}(A,B,C,D,E,nmeas,ncon,tol)$$

transform the generalized plant

$$E\dot{x} = Ax + B \begin{bmatrix} w \\ u \end{bmatrix}$$
$$\begin{bmatrix} z \\ y \end{bmatrix} = Cx + D \begin{bmatrix} w \\ u \end{bmatrix}$$

where the dimension of y is **nmeas** and the dimension of u is **ncon**, into an equivalent generalized plant

$$\begin{aligned} \dot{e}x &= ax + b \begin{bmatrix} w \\ u \end{bmatrix} \\ \begin{bmatrix} z \\ y \end{bmatrix} &= cx + d \begin{bmatrix} w \\ u \end{bmatrix} \end{aligned}$$

with

$$d = \begin{bmatrix} d11 & d12 \\ d21 & d22 \end{bmatrix}$$

such that **d12** has full column rank and **d21** has full row rank. “Equivalent” means that the two plants have the same transfer matrices.

The optional tolerance parameter **tol** is used in the various rank tests. It has the default value **1e-12**.

In verbose mode the routine displays a relative error based on the largest of the differences of the frequency response matrices of the transformed and the original plant at the frequencies 1, 2, ..., 10.

Examples

In the Example section of the manual page for the Polynomial Toolbox command **dsshinf** the descriptor representation of a generalized plant is derived. When considering the subsystem

$$z_2 = c(1 + rs)u$$

two pseudo state variables are defined as $x_3 = u$, $x_4 = \dot{u}$, which leads to the descriptor equations

$$\begin{aligned}\dot{x}_3 &= x_4 \\ 0 &= -x_3 + u\end{aligned}$$

The output equation is rendered as

$$z_2 = c(1+rs)u = crx_4 + cu$$

The output equation, however, equally well could be chosen as

$$z_2 = c(1+rs)u = cx_3 + crx_4$$

This brings the generalized plant in the form

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_E \underbrace{\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_A \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}} + \underbrace{\begin{bmatrix} \sqrt{2} & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}}_B \underbrace{\begin{bmatrix} w \\ u \end{bmatrix}}$$

$$\underbrace{\begin{bmatrix} z_1 \\ z_2 \\ y \end{bmatrix}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & c & cr \\ -1 & 0 & 0 & 0 \end{bmatrix}}_C \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}} + \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ -1 & 0 \end{bmatrix}}_D \underbrace{\begin{bmatrix} w \\ u \end{bmatrix}}$$

For this plant we have

$$D_{12} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad D_{21} = -1$$

so that D_{12} does not have full rank. We apply `dssreg` to this plant for $c = 0.1$, $r = 0.1$.

```
c = 0.1; r = 0.1;
E = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 0];
A = [0 1 0 0; 0 0 0 0; 0 0 0 1; 0 0 -1 0];
B = [sqrt(2) 0; 1 1; 0 0; 0 1];
C = [1 0 0 0; 0 0 c c*r; -1 0 0 0];
D = [1 0; 0 0; -1 0];
ncon = 1; nmeas = 1;
```

We now apply `dssreg`.

```
[a,b,c,d,e] = dssreg(A,B,C,D,E,nmeas,ncon)
```

```
a =
```

```
0      1      0      0
0      0      0      0
```

```

      0      0      0      1
      0      0     -1      0

b =
      1.4142      0
      1.0000      1.0000
           0      1.0000
           0      1.0000

c =
      1.0000      0      0      0
           0      0      0.1000      0.0100
     -1.0000      0      0      0

d =
      1.0000      0
           0      0.0100
     -1.0000      0

e =

```

$$\begin{bmatrix}
 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0
 \end{bmatrix}$$

We now have

$$D_{12} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad D_{21} = -1$$

so that the transformed plant is “regular.”

As a second example we consider the standard plant

$$\begin{aligned}
 \dot{x} &= x + \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} w \\ u \end{bmatrix} \\
 \begin{bmatrix} z \\ y \end{bmatrix} &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} x
 \end{aligned}$$

for which neither D_{12} nor D_{21} has full rank. We obtain the following result.

```

E = 1; A = 1; B = [1 1]; C = [1; 1]; D = [0 0; 0 0];
nmeas = 1; ncon = 1;
[a,b,c,d,e] = dssreg(A,B,C,D,E,nmeas,ncon)

```

a =

1	0	0
0	1	0
0	0	1

b =

1	1
0	1
1	0

c =

1	1	0
1	0	1

d =

0	1
1	0

e =

1	0	0
---	---	---

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Instead of a state representation of dimension 1 we now have a 3-dimensional descriptor representation, which, however, is “regular.”

Algorithm

First consider the case that D_{21} does not have full row rank.

Let the rows of the matrix N span the left null space of E , so that $NE = 0$. Then by multiplying the descriptor equation $E\dot{x} = Ax + B_1w + B_2u$ on the left by N we obtain the set of algebraic equations $0 = NAx + NB_1w + NB_2u$. By adding suitable linear combinations of the rows of this set of equations to the rows of the output equation $y = C_2x + D_{21}w + D_{22}u$ the rank of D_{21} may be increased without increasing the dimension of the pseudo state x .

If after this operation D_{21} still does not have full row rank then we apply a suitable transformation to the output equation $y = C_2x + D_{21}w + D_{22}u$ so that it takes the form

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} C_{21} \\ C_{22} \end{bmatrix} x + \begin{bmatrix} D_{211} \\ 0 \end{bmatrix} w + \begin{bmatrix} D_{221} \\ D_{222} \end{bmatrix} u$$

where D_{211} has full row rank. It is easy to construct a matrix D_{212} so that

$$\begin{bmatrix} D_{211} \\ D_{212} \end{bmatrix}$$

has full row rank. Following this we redefine the output equation as

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} C_{21} \\ C_{22} \end{bmatrix} x + \begin{bmatrix} 0 \\ I \end{bmatrix} x' + \begin{bmatrix} D_{211} \\ D_{212} \end{bmatrix} w + \begin{bmatrix} D_{221} \\ D_{222} \end{bmatrix} u$$

where x' is an additional component of the pseudo state. This component is accounted for by adding the algebraic equation

$$0 = x' + D_{212} w$$

to the descriptor equation. This increases the dimension of the pseudo state, of course.

If D_{12} does not have full rank then the procedure as described is applied to the “dual” system.

Diagnostics

The macro **dssreg** displays error messages in the following situations.

- The input parameters have inconsistent dimensions.
- D_{12} is not tall or D_{21} is not wide.
- The relative error exceeds $1\text{e-}6$. The relative error is computed on the basis of the largest of the differences of the frequency responses of the system before and after regularization at the frequencies 1, 2, ..., 10.

In verbose mode the relative error is always reported.

See also

<code>dssrch</code>	H_∞ optimization for a descriptor plant
<code>dssmin</code>	dimension reduction of a descriptor system

echelon

Purpose Echelon (or Popov) form of a polynomial matrix

Syntax

```
[E,U,ind] = echelon(D[,tol])
[E,U,ind] = echelon(D,'col'[,tol])
[E,U,ind] = echelon(D,'row'[,tol])
```

Description A polynomial matrix E is in column echelon form (or Popov form) if it has the following properties (Kailath, 1980).

- It is column reduced with its column degrees arranged in ascending order.
- For each column there is a so-called *pivot index* i such that the degree of the i th entry in this column equals the column degree, and the i th entry is the lowest entry in this column with this degree.
- The pivot indexes are arranged to be increasing.
- Each pivot entry is monic and has the highest degree in its row.

A square matrix in column echelon form is both column and row reduced.

Given a square and column-reduced polynomial matrix D the commands

```
[E,U,ind] = echelon(D)
```

```
[E,U,ind] = echelon(D,'col')
```

compute the column echelon form E of D . The unimodular matrix U satisfies $DU = E$. The first column of **ind** contains the pivot indices of E , the second column of **ind** contains its column degrees, and the third column of **ind** contains its row degrees.

Similarly, if D is row reduced then

```
[E,U,ind] = echelon(D,'row')
```

returns the row echelon form E of D while $UD = E$. A polynomial matrix is in row echelon form if it satisfies the properties listed above with 'column' replaced by 'row', and vice-versa.

A tolerance **tol** may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Consider the polynomial matrix

```
D = [ -3*s  s+2  
      1-s  1  ];
```

To find its column Hermite form and associated quantities we type

```
[E,U,ind] = echelon(D)
```

MATLAB returns

E =

$$\begin{bmatrix} s^2 + s & -6 \\ 1 & -4 + s \end{bmatrix}$$

Constant polynomial matrix: 2-by-2

U =

$$\begin{bmatrix} 0 & -1 \\ 1 & -3 \end{bmatrix}$$

ind =

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}$$

A more complicated example is

$$D = \begin{bmatrix} 5s+1 & s^2+3s+2 & 2s^3+3s^2+4s+5 \\ 3s+4 & 2s+1 & s^3+s^2+2 \\ s+7 & 3+4s & 2+5s+3s^2 \end{bmatrix};$$

Its echelon form is

$$E = \text{echelon}(D)$$

$E =$

$$\begin{bmatrix} 1 + 5s & -2 - 17s + s^2 & 15 + 2.1e+002s \\ 4 + 3s & -15 - 10s & 2e+002 + 1.4e+002s + 12s^2 + s^3 \\ 7 + s & -25 & 3.1e+002 \end{bmatrix}$$

Algorithm

The algorithm follows Kailath (1980, pp. 485–486). From $E = DU$ it follows that

$$\begin{bmatrix} D & -I \end{bmatrix} \begin{bmatrix} U \\ E \end{bmatrix} = 0$$

A polynomial basis for the right kernel of $[D - I]$ is built by application of the column-searching algorithm to the Sylvester resultant of this matrix. If only the “primary” dependent columns are selected then the set of columns of $[U; E]$ is of minimal degree and in polynomial column echelon form.

For the row echelon form D is required to be row reduced and the algorithm is applied to the transpose of D .

Diagnostics

The macro issues error messages when

- an invalid input argument or matrix argument is encountered
- invalid **Not-a-Number** or **Infinite** entries are found in the input matrices
- the input matrix is not square

- the input matrix is singular
- an invalid option is encountered
- the input matrix is not column or row reduced as required

See also

<code>tri</code>	triangular form of a polynomial matrix
<code>hermite</code>	Hermite form of a polynomial matrix
<code>smith</code>	Smith form of a polynomial matrix

eq, ne

Purpose Equality and inequality test for polynomial matrices $P == Q$, $P \sim Q$

Syntax

`A = (P==Q)`

`A = eq(P,Q)`

`A = eq(P,Q,tol)`

`A = (P~Q)`

`A = ne(P,Q)`

`A = ne(P,Q,tol)`

Description

The command

`A = (P==Q)`

performs an elementwise comparison between the polynomial matrices P and Q with tolerance activated through the global zeroing tolerance. P and Q must have the same dimensions unless one is a scalar polynomial. A scalar polynomial is compared with all entries.

A difference between the polynomial variables of P and Q causes a warning message, but does not affect the result. Thus, $(1+z) == (1+s)$ is 1.

The command

```
A = eq(P,Q)
```

works similarly. The command

```
A = eq(P,Q,tol)
```

works with the tolerance specified by the input tolerance `tol`.

The commands

```
A = (P~=Q)
```

```
A = ne(P,Q)
```

```
A = ne(P,Q,tol)
```

work similarly but return 0 where `eq` returns 1 and vice-versa.

Examples

Typing

```
P = [1+s 1+2*s];
```

```
1+s == P
```

results in

```
ans =
```

```
1      0
```

while


```
1+s/3 == 1+0.3333333333*s
```

returns

```
ans =
```

```
1
```

Likewise, typing

```
1+s ~= P
```

results in

```
ans =
```

```
0      1
```

while

```
1+s/3 ~= 1+0.3333333333*s
```

returns

```
ans =
```

```
0
```

Algorithm

The macros `eq` and `ne` use standard MATLAB 5 and Polynomial Toolbox operations.

Diagnostics

The macros `eq` and `ne` returns an error message if

- there are too few or too many input arguments
- there are too many output arguments
- the polynomial matrix dimensions do not agree

A warning is issued if the polynomial matrices have inconsistent variables.

evenpart, oddpart

Purpose Even and odd part of a polynomial

Syntax

```
[Aeven,Aodd] = evenpart(A)
[Aeven,Aodd] = evenpart(A,'sqz')
[Aodd,Aeven] = oddpart(A)
[Aodd,Aeven] = oddpart(A,'sqz')
```

Description For a polynomial matrix

$$A(s) = A_0 + A_1s + A_2s^2 + A_3s^3 + A_4s^4 + A_5s^5 + A_6s^6 + A_7s^7 + A_8s^8 + A_9s^9 + \dots$$

the command

```
evenpart(A)
```

returns its even part in the “full” form

$$A(s) = A_0 + A_2s^2 + A_4s^4 + A_6s^6 + A_8s^8 + \dots$$

while the command

```
evenpart(A,'sqz')
```

returns it in the “squeezed” form

$$A(s) = A_0 + A_2s + A_4s^2 + A_6s^3 + A_8s^4 + \dots$$

Similarly, the command

oddpart(A)

returns the even part in the “full” form

$$A(s) = A_1s + A_3s^3 + A_5s^5 + A_7s^7 + A_9s^9 + \dots$$

while the command

evenpart(A, 'sqz')

“squeezes” it to

$$A(s) = A_1 + A_3s + A_5s^2 + A_7s^3 + A_9s^4 + \dots$$

If either case is used with two output arguments, the other part is also returned in the same form.

Example

For illustration, consider

```
A=mono(0:9)*[-1 1:9].'
```

```
A =
```

```
-1 + s + 2s^2 + 3s^3 + 4s^4 + 5s^5 + 6s^6 + 7s^7 + 8s^8 +  
9s^9
```

and check that

```
evenpart(A)
ans =
    -1 + 2s^2 + 4s^4 + 6s^6 + 8s^8
evenpart(A,'sqz')
ans =
    -1 + 2s + 4s^2 + 6s^3 + 8s^4
oddpart(A)
ans =
    s + 3s^3 + 5s^5 + 7s^7 + 9s^9
oddpart(A,'sqz')
ans =
    1 + 3s + 5s^2 + 7s^3 + 9s^4
```

Algorithm

The routine uses standard routines from the Polynomial Toolbox.

Diagnostics

The macro returns error messages in the following situations:

- The first input argument is not a polynomial object

- The input string is incorrect

See also

`pol` polynomial matrix constructor

`{}`, `subsref` curly brackets operator and subscripted reference

fact

Purpose Polynomial matrix factor extraction

Syntax `AR = fact(A,z)`

`AR = fact(A,z,tol)`

Description Given a non-singular polynomial matrix A and a vector z containing a subset of the zeros of A , the instruction

`AR = fact(A,z)`

extracts from A a minimum degree, column and row reduced right polynomial matrix factor AR such that z contains all the zeros of AR and

$A = AL * AR$

The left factor AL can be retrieved with the instruction `AL = A/AR`.

An optional tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples Consider the polynomial matrix

$$A = \begin{bmatrix} 16-4*s+4*s^2 & 9-4*s+3*s^2 \\ 36-16*s+12*s^2 & 21-14*s+9*s^2 \end{bmatrix};$$

The matrix has the roots

```
roots(A)
```

```
ans =
```

```
-3.0000
```

```
1.0000
```

```
1.0000
```

We extract a right factor corresponding to the double root at 1 with the help of the command

```
AR = fact(A,[1 1])
```

```
AR =
```

```
1 - 2s + s^2    0
```

```
2              1
```

The left factor is

```
AL = A/AR
```

```
AL =
```

```
-2      9 - 4s + 3s^2
```


$$-6 \quad 21 - 14s + 9s^2$$

Algorithm

The algorithm is based on interpolation and relies on numerically reliable subroutines only. It makes use of the macro **character** to compute the generalized characteristic vectors of the polynomial matrix.

Diagnostics

The macro returns an error message in the following circumstances

- The input matrix is not square
- The second input argument is missing
- An invalid second input argument is encountered
- The input matrix is singular
- An invalid zero characteristic vector is obtained

See also

character compute the generalized characteristic vectors corresponding to a given root

spcof, **spf** spectral (co-)factorization

fliplr, flipud

Purpose Flip polynomial matrices left-right or up-down

Syntax `B = fliplr(A)`

`B = flipud(A)`

Description The command

`B = fliplr(A)`

returns the polynomial matrix A with its columns flipped in the left-right direction, that is, about a vertical axis. The command

`B = flipud(A)`

returns the polynomial matrix A with its columns flipped in the up-down direction, that is, about a horizontal axis.

Examples The commands

```
P = [ 1+s      2*s^2      4
      -2      3+4*s^3    5+6*s ];

Q = fliplr(P)
```

produce the response

$$Q = \begin{bmatrix} 4 & 2s^2 & 1 + s \\ 5 + 6s & 3 + 4s^3 & -2 \end{bmatrix}$$

while the commands

```
P = [ 1+s    2*s^2
      -2    3+4*s^3
      5      6+7*s ];
Q = flipud(P)
```

produce the response

$$Q = \begin{bmatrix} 5 & 6 + 7s \\ -2 & 3 + 4s^3 \\ 1 + s & 2s^2 \end{bmatrix}$$

Algorithm

The routines use standard routines from the Polynomial Toolbox.

Diagnostics

The macros display error messages if they have too many input or output arguments.

See also

`rot90`

rotate a polynomial matrix by 90 degrees

gld, grd**Purpose**

Greatest left and right divisor

Syntax

```

G = gld(A1,A2,...,Ak[,tol])

G = gld(A1,A2,...,Ak,'gau'[,tol])

G = gld(A1,A2,...,Ak,'syl'[,tol])

[G,U] = gld(A1,A2,...,Ak,'syl'[,tol])

G = grd(A1,A2,...,Ak[,tol])

G = grd(A1,A2,...,Ak,'gau'[,tol])

G = grd(A1,A2,...,Ak,'syl'[,tol])

[G,U] = grd(A1,A2,...,Ak,'syl'[,tol])

```

Description

The command

$$G = \text{gld}(N_1, N_2, \dots, N_k)$$

computes a greatest left divisor G of several polynomial matrices N_1, N_2, \dots, N_k (for $k > 0$) that all have the same number of rows. The columns of G form a polynomial basis for the module spanned by the columns of $N = [N_1 \ N_2 \ \dots \ N_k]$. Note that this basis is not necessarily of minimal degree. The number of rows in G equals the rank of N . If N has full row rank then G is square.

The command

```
G = gld(N1,N2,...,Nk,'gau')
```

computes the divisor through a modified version of Gaussian elimination. This method is preferable esthetically and generally results in a low degree divisor. It is the default method.

The command

```
G = gld(N1,N2,...,Nk,'syl')
```

computes the divisor through stable reductions of Sylvester matrices. This method is preferable numerically but may result in a high degree divisor. The command

```
[G,U] = gld(N1,N2,...,Nk,'syl')
```

in addition returns a unimodular matrix U such that

$$NU = \begin{bmatrix} G & 0 & \cdots & 0 \end{bmatrix}$$

Note that the second output argument is only allowed if the 'syl' option is used.

For two input matrices $\mathbf{N1}$ and $\mathbf{N2}$ the matrix U may be split into two pairs of right coprime polynomial matrices (P, Q) and (R, S) such that

$$U = \begin{bmatrix} P & R \\ Q & S \end{bmatrix}$$

so that

$$N1*P + N2*Q = G$$

$$N1*R + N2*S = 0$$

The matrices M_i such that $N_i = G*M_i$ may be recovered with the help of the command

$$M_i = \text{axb}(G, N_i)$$

A tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Similarly to `gld` the commands

$$G = \text{grd}(N1, N2, \dots, Nk)$$

$$G = \text{grd}(N1, N2, \dots, Nk, 'gau')$$

compute a greatest right common divisor G of several polynomial matrices $N1, N2, \dots, Nk$ (with $k > 0$) that all have the same column dimension. The rows of G form a polynomial basis for the module spanned by the rows of $N = [N1' \ N2' \ \dots \ Nk']'$. This basis is not necessarily of minimal degree. The number of rows in G is equal to the rank of N . If N has full column rank then G is square.

The command

$$[G, U] = \text{GRD}(N1, N2, \dots, Nk, 'syl')$$

additionally returns a unimodular matrix U such that

$$NU = \begin{bmatrix} G \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

For two input matrices **n1** and **n2** the matrix *U* may be split into two pairs of left coprime polynomial matrices (*P*, *Q*) and (*R*, *S*) such that

$$U = \begin{bmatrix} P & R \\ Q & S \end{bmatrix}$$

and hence

$$P^* N1 + Q^* N2 = G$$

$$R^* N1 + S^* N2 = 0$$

The matrices **mi** such that **ni = mi * G** may be recovered with the command

```
mi = xab(G, ni)
```

Examples

Let

```
A1 = [ s-s^2  
      2*s-s^2 ];  
A2 = [ -1+s      -1+s  
      -1+s      -2+s ];
```


A common left divisor follows as

```
G1 = gld(A1,A2)
```

```
G1 =
```

```
    -1 + s    0
```

```
    0         1
```

By application of

```
F1 = axb(G1,A1), F2 = axb(G1,A2)
```

the factors that remain after division follow as

```
F1 =
```

```
    -s
```

```
    2s - s^2
```

```
F2 =
```

```
    1         1
```

```
    -1 + s    -2 + s
```

The Sylvester method results in the left common divisor

```
[G2,U] = gld(A1,A2,'syl')
```

$$\begin{aligned}
 \mathbf{G2} &= \\
 &\begin{bmatrix} 1.4 & -1.4s & 0 \\ 2.1 & -1.4s & 0.71 \end{bmatrix} \\
 \mathbf{U} &= \\
 &\begin{bmatrix} 0 & 0 & 0.71 \\ -0.71 & 0.71 & 0 \\ -0.71 & -0.71 & 0.71s \end{bmatrix}
 \end{aligned}$$

$\mathbf{G1}$ and $\mathbf{G2}$ are equivalent within elementary row operations.

Algorithm

The routine `gld` is the dual of `grd`.

The algorithm '`gau`' of `grd` relies on Gaussian elimination of the resultant matrix of the polynomial matrix $A = [\mathbf{A1}; \mathbf{A2}; \dots, \mathbf{Ak}]$ as described by Barnett (1983).

The algorithm '`sy1`' relies on transforming A to lower triangular form. This is done in a numerically reliable way using the macro `echelon`.

Diagnostics

The macros issue error messages when

- an invalid input argument is encountered
- invalid `Not-a-Number` or `Infinite` entries are found in the input matrices

- the input matrices have inconsistent dimensions
- the input matrices have inconsistent variables
- an invalid method is encountered

See also

<code>axb, xab</code>	linear polynomial matrix equation solvers
<code>ldiv, rdiv</code>	left and right polynomial matrix division
<code>minbasis</code>	minimal basis
<code>echelon, hermite</code>	echelon and Hermite form of polynomial matrices
<code>llm, rlm</code>	least multiples of polynomial matrices

gram

Purpose Gramian of a polynomial matrix fraction

Syntax

```
G = gram(N,D[,tol])
G = gram(N,D,'l'[,tol])
G = gram(N,D,'r'[,tol])
```

Description The commands

```
G = gram(N,D)
```

```
G = gram(N,D,'l')
```

result in the computation of the Gramian of the stable coprime polynomial matrix fraction $H = D^{-1}N$. In the continuous-time case the Gramian is defined as

$$G = \frac{1}{2p} \int_{-\infty}^{\infty} H^T(-jw)H(jw) dw$$

and in the discrete-time case as

$$G = \frac{1}{2p} \int_{-p}^p H^T(e^{-jw})H(e^{jw}) dw$$

The command

```
G = gram(N,D,'r')
```

computes the Gramian of the stable rational matrix $H = ND^{-1}$ with N and D right coprime.

A tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

The controllability and observability Gramians of a state space system with matrices A , B , C , D may be computed as

```
Wc = gram(B,v*I-A,'l')
```

```
Wo = gram(C,v*I-A,'r')
```

where v is the variable and I stands for the identity matrix of the correct dimension.

Examples

Consider the state space system defined by

```
A = [ 0  1
      -3 -4 ];

B = [ 1
      1 ];

C = [ 1  0 ];
```

The controllability Gramian follows as

```
Wc = gram(B,s*eye(2)-A,'l')
```

```
Wc =
```

```
1.1667    -0.5000
-0.5000    0.5000
```

while the observability Gramian is given by

```
Wo = gram(C,s*eye(2)-A,'r')
```

```
Wo =
```

```
0.7917    0.1667
0.1667    0.0417
```

As a discrete-time example, consider

```
N = 1; D = z-0.5;
```

We then find

```
G = gram(N,D)
```

```
G =
```

```
1.3333
```

Algorithm	For the continuous-time case the algorithm relies on the solution of a two-sided polynomial matrix equation as described in Kwakernaak (1992). For the discrete-time case a similar solution may be developed.
Diagnostics	<p>The macro <code>dssmin</code> displays error messages in the following situations</p> <ul style="list-style-type: none"> • There are not enough input arguments • An unknown command option is encountered • The input matrices have inconsistent dimensions • The input matrices are not coprime • The denominator matrix is not stable • The computation of the Gramian failed <p>A warning follows if the input matrices have inconsistent variables. In this case both variables are set equal to the default global variable.</p>
See also	<code>h2norm</code> computation of the H_2 norm of a matrix fraction

h2norm, hinfnorm

Purpose H_2 norm of a polynomial matrix fraction
 H_∞ norm of a polynomial matrix fractions

Syntax

```
n = h2norm(N,D[,tol])
n = h2norm(N,D,'l'[,tol])
n = h2norm(N,D,'r'[,tol])

[ninf,omegamax] = hinfnorm(N,D)
[ninf,omegamax] = hinfnorm(N,D,'l')
[ninf,omegamax] = hinfnorm(N,D,'r')
```

Description The commands

```
n2 = h2norm(N,D)
n2 = h2norm(N,D,'l')
```

return the H_2 norm of the stable coprime polynomial fraction $G = D^{-1}N$, while

```
n2 = h2norm(N,D,'r')
```


returns the H_2 norm of $G = ND^{-1}$. If N and D have s or p as variable then the continuous-time norm is used, if it is z , d , q or z^{-1} then the discrete-time norm applies.

A tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

The commands

```
ninf = hinfnorm(N,D)
```

```
ninf = hinfnorm(N,D,'l')
```

return the H_∞ norm of the stable rational matrix $G = D^{-1}N$, while

```
ninf = hinfnorm(N,D,'r')
```

computes the H_∞ norm of $G = ND^{-1}$. The command

```
[ninf,omegamax] = hinfnorm(N,D)
```

possibly combined with the 'l' or 'r' option, also returns the frequency `omegamax` where $\|G(j\omega)\|_2$ achieves its maximum. The norm is computed in continuous or discrete time depending on the variable of N and D .

Examples

Define the polynomial matrices

```
D = [ s+1  2
      1  s+4 ];
```

```
N = eye(2);
```

Then

```
h2norm(N,D)
```

returns

```
ans =
```

```
1.1726
```

while

```
[ninf,omegamax] = hinfnorm(N,D)
```

returns

```
ninf =
```

```
2.3354
```

```
omegamax =
```

```
0
```

For the discrete-time system defined by

```
N = 1; D = z-0.5;
```

the norms are computed as

```
h2norm(N,D)
```

```
ans =
```

```
1.1547
```

and

```
hinfnorm(N,D)
```

```
ans =
```

```
2
```

Algorithm

The algorithm for the computation of the H_2 norm relies on the computation of the Gramian of the matrix fraction (Kwakernaak, 1992).

The H_∞ norm is computed by searching for the peak value of

$$\|G(jw)\|_2, \quad w \in [0, \infty], \quad \text{or} \quad \|G(e^{jw})\|_2, \quad w \in [0, p],$$

first on a coarse grid, which then is refined.

Diagnostics

The macros returns error messages in the following circumstances:

- There are not enough input arguments or too many output arguments
- An unknown command option is encountered
- The input matrices have inconsistent dimensions

- (H_2 case only) The computation of the Gramian failed

Warnings follow if

- The denominator matrix is not stable so that the H_2 or H_∞ norm is infinite
- (H_2 case only) The matrix fraction is not strictly proper, so that the H_2 norm is infinite
- (H_∞ case only) D has a pole on the imaginary axis so that the norm is infinite
- (H_∞ case only) The fraction is nonproper so that the norm is infinite
- (H_∞ case only) The denominator is zero to working precision

See also

norm	computation of various norms
gram	computation of the Gramian

hermite

Purpose Hermite form of a polynomial matrix

Syntax

```
[H,U,ind] = hermite(A[,tol])
[H,U,ind] = hermite(A,'col'[,tol])
[H,U,ind] = hermite(A,'row'[,tol])
```

Description An $n \times m$ polynomial matrix A of rank r is in column Hermite form if it has the following properties.

- It is lower triangular
- The diagonal entries are all monic
- Each diagonal entry has higher degree than any entry on its left
- In particular, if the diagonal element is constant then all off-diagonal elements in the same row are zero
- If $n > r$ then the last $n-r$ columns are zero

The nomenclature in the literature is not consistent. Some authors refer to this as the row Hermite form.

The polynomial matrix A is in row Hermite form if it is the transpose of a matrix in column Hermite form.

The commands

```
H = hermite(A)
```

```
H = hermite(A, 'col')
```

compute the column Hermite form of A . The command

```
[H,U,ind] = hermite(A)
```

also provides a unimodular reduction matrix U such that $AU = H$ and a row vector **ind** such that **ind**(i) is the row index of the uppermost non-zero entry in the i th column of H . Note that **ind**(i) = 0 if the i th column of H is zero.

The command

```
hermite(A, 'row')
```

computes the row Hermite form of A .

The macro **hermite** is partly based on macro **tri**. The optional input arguments '**gau**' and '**syl**' for **tri** may be specified through **hermite**.

An optional tolerance **tol** may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Consider the polynomial matrix

```
A = [ s^2    0    1
      0    s^2  1+s];
```

To find its column Hermite form and associated quantities we type

```
[H,U,ind] = hermite(A)
```

MATLAB returns

H =

```

1      0      0
1 + s  s^2    0

```

U =

```

0      -0.25      0.5
0      0.75 - 0.25s  0.5 + 0.5s
1      0.25s^2      -0.5s^2

```

ind =

```

1      2      0

```

For the row Hermite form the command

```
hermite(A,'row')
```

returns the original matrix

ans =

$$\begin{array}{ccc} s^2 & 0 & 1 \\ 0 & s^2 & 1 + s \end{array}$$

Algorithm

The macro `hermite` is based on `tri`. First, `tri` is called to compute a staircase form. Second, the off-diagonal terms are reduced with the macro `rdiv`. Finally, the diagonal terms are made monic.

Diagnostics

The macro `hermite` issues error messages if

- an invalid input argument or matrix argument is encountered
- invalid `Not-a-Number` or `Infinite` entries are found in the input matrices
- the reduction fails in one of several ways

In the latter case the tolerance should be modified.

See also

`tri` triangular form of a polynomial matrix
`smith` Smith form of a polynomial matrix
`lu` LU factorization for polynomial matrices

horzcat, vertcat

Purpose Horizontal and vertical concatenation for polynomial matrix objects

Syntax

$$C = [A \ B]$$

$$C = [A, B]$$

$$C = \text{horzcat}(A, B)$$

$$C = [A$$

$$\quad B]$$

$$C = [A; B]$$

$$C = \text{vertcat}(A, B)$$
Description

The command

$$C = [A \ B]$$

effects the horizontal concatenation of the polynomial matrices A and B . A and B must have the same number of rows. The commands

$$C = [A, B]$$

and

```
C = horzcat(A,B)
```

have the same result. Any number of matrices can be concatenated within one pair of brackets.

The command

```
C = [A
     B]
```

effects the horizontal concatenation of the polynomial matrices A and B . A and B must have the same number of columns. The commands

```
C = [A,B]
```

and

```
C = vertcat(A,B)
```

have the same result. Any number of matrices can be concatenated within one pair of brackets.

Horizontal and vertical concatenation can be combined together as for standard MATLAB matrices.

Examples

Typing

```
P = [1+s 1+2*s]
```

results in

```

P =
      1 + s      1 + 2s
Q = [P; 0 1]
Q =
      1 + s      1 + 2s
      0          1

```

Algorithm

The macros use standard MATLAB 5 and Polynomial Toolbox operations.

Diagnostics

The macros `horzcat` and `vertcat` return an error message if not all matrices on a row or in a column in the bracketed expression have the same number of rows or columns.

hurwitz

Purpose Create the Hurwitz matrix of a polynomial matrix

Syntax `H = hurwitz(P)`

`H = hurwitz(P,k)`

Description Given a polynomial matrix P the command

`H = hurwitz(P,k)`

creates the constant Hurwitz matrix H of order k ($k \times k$ blocks) corresponding to the polynomial matrix P . If

$$P(v) = P_0 + P_1 v + P_2 v^2 + \cdots + P_d v^d$$

then

$$H = \begin{bmatrix} P_{d-1} & P_{d-3} & P_{d-5} & \cdots & \cdots & \cdots & \cdots \\ P_d & P_{d-2} & P_{d-4} & \cdots & \cdots & \cdots & \cdots \\ 0 & P_{d-1} & P_{d-3} & P_{d-5} & \cdots & \cdots & \cdots \\ 0 & P_d & P_{d-2} & P_{d-4} & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & P_{d-1} & P_{d-3} & P_{d-5} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{matrix} \text{block row 1} \\ \text{block row 2} \\ \text{block row 3} \\ \vdots \\ \vdots \\ \vdots \\ \text{block row } k \end{matrix}$$

The default value of k is d .

Examples

Consider the polynomial matrix

```
P = [1+s 2*s^2 3];
```

Then the command

```
H = hurwitz(P)
```

results in

```
H =
```

```

      1      0      0      0      0      0
      0      2      0      1      0      3

```

If

```
P = mono(0:3)*[1; 2; 3; 4]
```

```
P =
```

```

      1 + 2s + 3s^2 + 4s^3

```

then

```
H = hurwitz(P)
```

returns

```
H =  
  
    3    1    0  
    4    2    0  
    0    3    1
```

Algorithm The macro uses basic operations from MATLAB 5 and the Polynomial Toolbox.

Diagnostics The macro returns an error message if the first input argument is not a polynomial object, or if the requested order is not a nonnegative integer.

See also `stabint` stability interval test

imag

Purpose Imaginary part of a polynomial matrix

Syntax `x = imag(Z)`

Description For a polynomial matrix Z , the command `x = imag(Z)` returns the polynomial matrix X whose coefficients are the imaginary parts of the corresponding coefficients of Z .

Examples If

```
P = [(1+i)*s (1-i)*s+s^2];
```

then

```
imag(P)
```

results in

```
ans =  
  
s      -s
```

Algorithm The macro `imag` uses standard MATLAB operations.

Diagnostics The macro displays no error messages.

See also `real` real part of a polynomial matrix

<code>ctranspose (')</code>	complex conjugate transpose of a polynomial matrix
<code>conj</code>	conjugate of a polynomial matrix

inertia

Purpose Inertia of a polynomial matrix

Syntax `in = inertia(A)`
`in = inertia(A,tol)`

Description The commands

```
in = inertia(A)
in = inertia(A,tol)
```

return the inertia of the polynomial matrix A . The inertia is the triple `in = [ns n0 nu]`, with `ns` the number of stable roots of A , `n0` the number of marginally stable roots, and `nu` the number of unstable roots.

The definition of stability depends on the polynomial matrix indeterminate. The root r of a polynomial matrix is considered as stable

- with indeterminate s or p if `real(r) < 0`
- with indeterminate z^{-1} or d if `abs(r) > 1`
- with indeterminate z or q if `abs(r) < 1`

The optional input parameter `tol` is a tolerance which is used to determine if certain coefficients that arise in the construction of the Routh array are zero. Its default value is the global zeroing tolerance.

Example

We consider the polynomial matrix

```
P = [1+s 2+s^2 3; 4-s 5 6+s^2; 1 1 1]
```

```
P =
```

```

      1 + s      2 + s^2      3
      4 - s      5          6 + s^2
      1          1          1

```

We compute the roots and the inertia of P

```
rtsP = roots(P), inP = inertia(P)
```

```
rtsP =
```

```

      0
-0.2980 + 1.8073i
-0.2980 - 1.8073i
      0.5961

```

```
inP =
```

2 1 1

Algorithm

The algorithm and its code are from Meinsma (1995).

Diagnostics

The macro issues an error message if the input matrix is not square.

See also

`isstable` stability of a polynomial matrix

isempty, isfinite, isinf, isnan, isreal**Purpose** Detect state

Syntax

```
s = isempty(P)
```

```
s = isfinite(P)
```

```
s = isinf(P)
```

```
s = isnan(P)
```

```
s = isreal(P)
```

Description The command

```
s = isempty(P)
```

returns the logical true (1) if P is an empty polynomial matrix and the logical false (0) otherwise. An empty polynomial matrix has at least one dimension of size zero, for instance, 0×0 or 0×5 .

The command

```
s = isfinite(P)
```

returns an array S of the same size as P containing the logical true (1) if the elements of the polynomial matrix have all coefficients finite and the logical false (0) if they have infinite or **NaN** coefficients.

The command

```
S = isinf(P)
```

returns an array S of the same size as P containing the logical true (1) if any of the coefficients of the corresponding entry of P is **+Inf** or **-Inf** and the logical false (0) if it is not.

The command

```
S = isnan(P)
```

returns an array S of the same size as P containing the logical true (1) if any of the coefficients of the corresponding entry of P is **NaN** and the logical false (0) if it is not.

The command

```
s = isreal(P)
```

returns the logical true (1) if all the entries of P have real coefficients and the logical false (0) otherwise.

Examples

We illustrate the various commands without further comments.

- **isempty**

```
isempty(pol(zeros(0,3)))
```

```
ans =
```

1

▪ `isfinite``P = [1+s Inf]; isfinite(P)``ans =`

1 0

▪ `isinf``isinf(P)``ans =`

0 1

▪ `isnan``P = [1+s 0/0]; isnan(P)``Warning: Divide by zero.``ans =`

0 1

▪ `isreal``isreal(P)`

```
ans =
```

```
1
```

Algorithm The macros rely on standard MATLAB 5 and Polynomial Toolbox operations.

Diagnostics The macros display no error messages.

See also `real`, `imag` real or imaginary part of a polynomial matrix

isfullrank, isprime, isproper, issingular, isstable, isunimod

Purpose Detect state

Syntax

```
s = isfullrank(P[,tol])  
  
[s,rts] = isprime(P[,tol])  
  
[s,rts] = isprime(P,'l'[,tol])  
  
[s,rts] = isprime(P,'r'[,tol])  
  
s = isproper(N,D[,tol])  
  
s = isproper(N,D,'l'[,tol])  
  
s = isproper(N,D,'r'[,tol])  
  
s = isproper(N,D,'strict'[,tol])  
  
s = isproper(N,D,'l','strict'[,tol])  
  
s = isproper(N,D,'r','strict'[,tol])  
  
s = issingular(P[,tol])  
  
s = isstable(P[,tol])  
  
s = isunimod(P[,tol])
```


Description

Isfullrank

The command

```
s = isfullrank(P)
```

returns the logical true (1) if P has full rank and the logical false (0) otherwise. A polynomial matrix has full rank if $\text{rank}(P) = \min(\text{size}(P))$.

The optional tolerance `tol` is interpreted as in the macro **rank**.

Isprime

The commands

```
isprime(P)
```

```
isprime(P, 'l')
```

return 1 if the polynomial matrix P is left prime and 0 if it is not. A polynomial matrix $P(s)$ is left prime if it has full row rank for all s in the complex plane, and right prime if it has full column rank. In the form

```
isprime(P, 'r')
```

the command tests whether the matrix is right prime. In the form

```
[r, rts] = isprime(P)
```

```
[r, rts] = isprime(P, 'l')
```

```
[r,rts] = isprime(P,'r')
```

the optional output argument **rts** returns the roots of P , that is, those points in the complex plane where P loses rank:

- If P is prime then **rts** is empty.
- If P is tested to be not left prime then **rts** contains those points where P loses row rank. If **rts** is returned as **NaN** then P does not have full row rank.
- If P is tested to be not right prime then **rts** contains those points where P loses column rank. If **rts** is returned as **NaN** then P does not have full column rank.

If P is square and prime then P is unimodular.

As last input argument an optional tolerance **tol** may be included that is used to test whether P has any roots. Its default value is the global zeroing tolerance.

Isproper

A polynomial matrix fraction $H = D^{-1}N$ or $H = ND^{-1}$ in the variables s, p, z or q is proper if $H(\infty)$ is finite, and strictly proper if $H(\infty) = 0$. If the variable is d or z^{-1} then properness or strict properness depends on whether $H(0)$ is finite or zero.

Given two polynomial matrices N and D , the commands

```
isproper(N,D)
```

isproper(N,D,'l')

return 1 if the left matrix fraction $D^{-1}N$ is a proper rational function and 0 if it is not. Similarly,

isproper(N,D,'r')

returns 1 if the right matrix fraction ND^{-1} is a proper rational function, and 0 otherwise.

The command

isproper(N,D,'strict')

possibly combined with the 'l' or 'r' option, tests if the matrix fraction is strictly proper.

A tolerance **tol** may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Issingular

The command

s = issingular(P)

returns the logical true (1) if the square polynomial matrix P is singular and 0 otherwise. The optional tolerance **tol** is interpreted as in the macro **rank**.

Isstable

The command

```
s = isstable(P)
```

returns the logical true (1) if P is stable and the logical false (0) otherwise. The definition of stability depends on the polynomial matrix variable symbol. A polynomial matrix with roots \mathbf{z}_i is considered as stable

- for the variable symbols s or p : if $\text{real}(\mathbf{z}_i) < 0$
- for the variable symbols z^{-1} or d : if $\text{abs}(\mathbf{z}_i) > 1$
- for the variable symbols z or q : if $\text{abs}(\mathbf{z}_i) < 1$

for all i . It is considered unstable otherwise. If the matrix is constant then it is considered unstable.

A tolerance `tol` may be specified as an additional input argument. It is used in testing whether an element of the Routh array is nonpositive. Its default value is the global zeroing tolerance times $1e-8$.

Isunimod

The command

```
s = isunimod(P)
```

returns 1 if the polynomial matrix P is unimodular and 0 if it is not. An optional tolerance `tol` may be included. Its default value is the global zeroing tolerance.

Examples

Isfullrank

Define

```
P = [ 1  s
      1  s ];
```

Obviously

```
isfullrank(P)
```

returns

```
ans =
     0
```

Isprime

The command

```
[r,rts] = isprime(P)
```

returns

```
r =
```

```

0
rts =
NaN

```

because P does not have full rank. Define

```
D = (1+s)^2; N = 1+s;
```

Then

```
[r,rts] = isprime([D N])
```

correctly returns

```

r =
0
rts =
-1.0000

```

Isproper

Define

```

D = (1+s)^2;
N = [ 2+s s^2 ];

```

Then

```
isproper(N,D)
```

returns

```
ans =  
  
1
```

while

```
isproper(N(1,1),D,'strict')
```

also returns

```
ans =  
  
1
```

Issingular

The command

```
issingular(P)
```

correctly yields

```
ans =  
  
1
```

Isstable

Given the polynomials

```
S = 2+s; Z = 2+z; Zi = 2+zi;
```

the stability checks successively return

```
isstable(S)
```

```
ans =
```

```
1
```

```
isstable(Z)
```

```
ans =
```

```
0
```

```
isstable(Zi)
```

```
ans =
```

```
1
```

Isunimod

Let

```
P = prand(2,2,2,'uni')
```



```
P =
      -0.59 + s + 0.59s^2      -1.7 - 0.61s
      0.96 - 0.97s           1
```

Then

```
isunimod(P)

ans =

      1
```

confirms that P is unimodular.

Algorithm

Isfullrank

The macro **isfullrank** calls **rank** to compute the rank of P .

Isprime

The $n \times m$ input argument P is “squared down” by postmultiplying it by an $m \times n$ random constant matrix W_1 and is further conditioned by premultiplying it by a square constant random matrix V_1 . Subsequently $d(s) = \det V_1 P(s) W_1$ is computed. Any value of s where $P(s)$ loses rank is a root of d . Which of the roots of d actually makes P lose rank is tested by checking which of the roots of d is also a root of $\det V_2 P(s) W_2$ with V_2 and W_2 further random matrices of the correct dimensions.

Right primeness of P is verified by testing P' for left primeness.

Isproper

If the variable is d or z^{-1} then the fraction is proper if $D(0)$ is nonsingular and strictly proper if $D(0)$ is nonsingular and $N(0) = 0$.

If the variable is s , p , z or q then first D is transformed to column or row reduced form as needed, and properness or strict properness is established by comparing the column or row degrees of N and D .

Isstable

Stability of the polynomial matrix P is tested by applying the continuous- or discrete-time Routh-Hurwitz test to $\det P$.

Isunimod

First the determinant is computed without zeroing. If the determinant is nonzero and finite and the constant coefficient matrix of the determinant is at least $1/\epsilon_0$ times greater than the other terms then the matrix is declared unimodular.

If the determinant is zero or infinite then the rank of the matrix is checked using reliable methods. The determinant is scale-sensitive and concluding the singularity from the zero determinant is numerically dangerous.

If the rank of the matrix is not full then the result of **isunimod** is 0 and a warning message saying “Matrix is singular to working precision” is displayed.

If the determinant is zero or infinite but the matrix appears to have full rank then the determinant has under- or overflowed. In this case an attempt is made to multiply the input matrix by a suitable constant matrix to make the leading or closing coefficient of the determinant equal to one. The coefficients of the determinant of the scaled matrix are then compared to establish unimodularity. If no suitable scaling matrix is found then the macro returns 0 and displays a warning message saying “Matrix is badly scaled.” This happens very rarely.

Diagnostics

The macro **isprime** complains if it encounters an unknown option, and **issingular**, **isstable** and **isunimod** if the input polynomial matrix is not square.

See also

rank	rank of a polynomial matrix
gld, grd	greatest left or right divisor

kharit

Purpose	Computation of Kharitonov polynomials and robust stability test for an interval polynomial
Syntax	$[\text{stab}, K1, K2, K3, K4] = \text{kharit}(\text{pmin}, \text{pmax})$ $[\text{stab}, K1p, K2p, K3p, K4p, K1n, K2n, K3n, K4n] = \text{kharit}(\text{pmin}, \text{pmax})$
Description	The scalar real interval polynomial

$$p(s, q) = \sum_{i=1}^n [q_i^-, q_i^+] s^i$$

is characterized by the two polynomials

$$p_{\min}(s) = \sum_{i=1}^n q_i^- s^i \quad \text{and} \quad p_{\max}(s) = \sum_{i=1}^n q_i^+ s^i$$

The command

```
[stab, K1, K2, K3, K4] = kharit(pmin, pmax)
```

computes the four Kharitonov polynomials

$$\begin{aligned}
 K_1(s) &= q_0^- + q_1^- s + q_2^+ s^2 + q_3^+ s^3 + q_4^- s^4 + q_5^- s^5 + q_6^- s^6 + \dots \\
 K_2(s) &= q_0^+ + q_1^+ s + q_2^- s^2 + q_3^- s^3 + q_4^+ s^4 + q_5^+ s^5 + q_6^- s^6 + \dots \\
 K_3(s) &= q_0^+ + q_1^- s + q_2^- s^2 + q_3^+ s^3 + q_4^+ s^4 + q_5^- s^5 + q_6^- s^6 + \dots \\
 K_4(s) &= q_0^- + q_1^+ s + q_2^+ s^2 + q_3^- s^3 + q_4^- s^4 + q_5^+ s^5 + q_6^+ s^6 + \dots
 \end{aligned}$$

If all the interval polynomial have invariant degree and all the Kharitonov polynomials are stable then $p(s, q)$ is robustly stable and **stab** is returned as 1.

If the interval polynomial has complex coefficients and is given by

$$p(s, q, r) = \sum_{i=1}^n ([q_i^-, q_i^+] + [r_i^-, r_i^+]) s^i$$

then it is characterized by two polynomials p_{\min} and p_{\max} with complex coefficients. The command

[stab, K1p, K2p, K3p, K4p, K1n, K2n, K3n, K4n] = kharit(pmin, pmax)

then returns the eight complex Kharitonov polynomials

$$\begin{aligned}
 K_1^+(s) &= (q_0^- + jr_0^-) + (q_1^- + jr_1^+)s + (q_2^+ + jr_2^+)s^2 + (q_3^+ + jr_3^-)s^3 + \dots \\
 K_2^+(s) &= (q_0^+ + jr_0^+) + (q_1^+ + jr_1^-)s + (q_2^- + jr_2^-)s^2 + (q_3^- + jr_3^+)s^3 + \dots \\
 K_3^+(s) &= (q_0^+ + jr_0^-) + (q_1^- + jr_1^-)s + (q_2^- + jr_2^+)s^2 + (q_3^+ + jr_3^+)s^3 + \dots \\
 K_4^+(s) &= (q_0^- + jr_0^+) + (q_1^+ + jr_1^+)s + (q_2^+ + jr_2^-)s^2 + (q_3^- + jr_3^-)s^3 + \dots
 \end{aligned}$$

$$\begin{aligned}
 K_1^-(s) &= (q_0^- + jr_0^-) + (q_1^+ + jr_1^-)s + (q_2^+ + jr_2^+)s^2 + (q_3^- + jr_3^+)s^3 + \dots \\
 K_2^-(s) &= (q_0^+ + jr_0^+) + (q_1^- + jr_1^+)s + (q_2^- + jr_2^-)s^2 + (q_3^+ + jr_3^-)s^3 + \dots \\
 K_3^-(s) &= (q_0^+ + jr_0^-) + (q_1^+ + jr_1^+)s + (q_2^- + jr_2^+)s^2 + (q_3^- + jr_3^-)s^3 + \dots \\
 K_4^-(s) &= (q_0^- + jr_0^+) + (q_1^- + jr_1^-)s + (q_2^+ + jr_2^-)s^2 + (q_3^+ + jr_3^+)s^3 + \dots
 \end{aligned}$$

If all the complex Kharitonov polynomials are stable and their degree is invariant then the complex interval polynomial $p(s, q, r)$ is robustly stable and **stab** is returned as 1.

If p is a matrix, then the macro works entry-wise.

Noce: For discrete-time polynomials, the function formally works as described above but robust stability cannot be concluded from the stability of the Kharitonov polynomials.

Example

The interval polynomial (Barmish 1994, p. 69)

$$p(s, q) = [0.25, 1.25] + [0.75, 1.25]s + [2.75, 3.25]s^2 + [0.25, 1.25]s^3,$$

is defined by its bounding polynomials

```

pmin = pol([0.25 0.75 2.75 0.25],3)

pmin =

    0.25 + 0.75s + 2.8s^2 + 0.25s^3
    
```

and

```
pmax = pol([1.25 1.25 3.25 1.25],3)
```

```
pmax =
```

```
1.3 + 1.3s + 3.3s^2 + 1.3s^3
```

Its robust stability is confirmed and the four Kharitonov polynomials are computed by

```
[stab,K1,K2,K3,K4] = kharit(pmin,pmax)
```

```
stab =
```

```
1
```

```
K1 =
```

```
0.25 + 0.75s + 3.3s^2 + 1.3s^3
```

```
K2 =
```

```
1.3 + 1.3s + 2.8s^2 + 0.25s^3
```

```
K3 =
```

```
1.3 + 0.75s + 2.8s^2 + 1.3s^3
```

```
K4 =
```

```
0.25 + 1.3s + 3.3s^2 + 0.25s^3
```

Algorithm

The routine uses standard routines from the Polynomial Toolbox.

Diagnostics

The macro returns error messages in the following situations:

- The input arguments are not polynomial objects
- The input matrices have inconsistent dimensions

It displays warning messages if

- The input polynomials are not of a continuous-time nature, that is, their variable is neither s nor p
- The interval polynomial is not degree invariant

See also

`khp1ot` plot Kharitonov rectangles

khplot

Purpose Plotting Kharitonov rectangles for an interval polynomial to test its robust stability by the Zero Exclusion Condition

Syntax `khplot(pmin,pmax,omega)`
`khplot(pmin,pmax,omega,'new')`

Description The scalar real interval polynomial

$$p(s,q) = \sum_{i=1}^n [q_i^-, q_i^+] s^i$$

is characterized by the two polynomials

$$p_{\min}(s) = \sum_{i=1}^n q_i^- s^i \text{ and } p_{\max}(s) = \sum_{i=1}^n q_i^+ s^i$$

Its uncertainty bounding set

$$Q = \left\{ q \mid q = (q_1, \dots, q_n), q_i \in [q_i^-, q_i^+], i = 1, 2, \dots, n \right\}$$

is a box in the space of parameters.

For a fixed frequency $w = w_0$, the command

`khplot(pmin,pmax,omega0)`

plots the set of all possible values that $p(jw_0, q)$ may assume as q ranges over the box Q , or more formally, the subset of complex plane given by

$$p(jw_0, Q) = \{p(jw_0, q) : q \in Q\}$$

This is called the Kharitonov rectangle at the frequency w_0 .

If `omega` is a vector of frequencies then the command

`khplot(pmin,pmax,omega)`

plots the Kharitonov rectangles “in motion.” This may be used a simple graphical check for robust stability of $p(s, q)$ based on the well-known Zero Exclusion Condition.

The command

`khplot(pmin,pmax,omega, 'new')`

opens a new figure window for the plot leaving any existing figure windows unchanged.

Noce: The macro plots a correct value set $p(jw, Q)$ only for *real* `omega` and hence it should only be used to test robust stability with the imaginary axis as a boundary. For the general case of complex frequencies, such as on the unit circle when testing for discrete-time robust stability, the resulting value set is not a rectangle and the more general macro `ptoplot` must be used.

Example 1

Motion of Kharitonov rectangle. The interval polynomial (Barmish 1994, p. 74)

$$p(s, q) = [0.25, 1.25] + [0.75, 1.25]s + [2.75, 3.25]s^2 + [0.25, 1.25]s^3,$$

is defined by its bounding polynomials

```
pmin = pol([0.25 0.75 2.75 0.25],3)
pmin =
0.25 + 0.75s + 2.8s^2 + 0.25s^3
```

and

```
pmax = pol([1.25 1.25 3.25 1.25],3)
pmax =
1.3 + 1.3s + 3.3s^2 + 1.3s^3
```

The motion of its Kharitonov rectangle $p(j\omega, Q)$ for twenty evenly spaced frequencies between $\omega = 0$ and $\omega = 1$ is produced by the command

```
khplot(pmin,pmax,0:.05:1)
```

It is shown in Fig. 3.

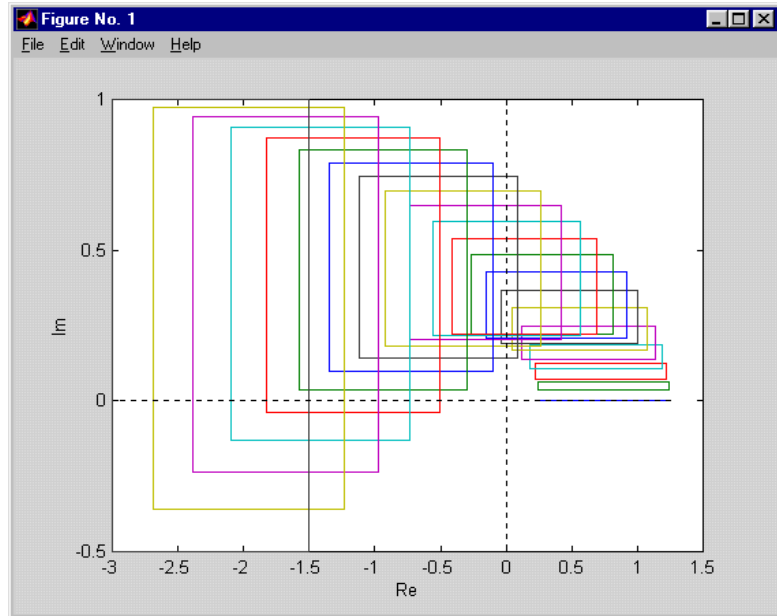


Fig. 3. Motion of the Kharitonov rectangles

Example 2

Robust stability test by the Zero Exclusion Condition. Consider the interval polynomial (Barmish 1994, p. 80)

$$p(jw, q) = [0.45, 0.55] + [1.95, 2.05]s + [2.95, 3.05]s^2 + [5.95, 6.05]s^3 \\ + [3.95, 4.05]s^4 + [3.95, 4.05]s^5 + s^6$$

The first step in the graphical test for robust stability requires that we guarantee that at least one polynomial in the family is stable. Using the midpoint of each interval we obtain

```
pp0 = pol([0.5 2 3 6 4 4 1],6)

pp0 =

    0.5 + 2s + 3s^2 + 6s^3 + 4s^4 + 4s^5 + s^6

isstable(pp0)

ans =

    1
```

The next step is to check whether for any w the associated Kharitonov rectangle includes the point $z = 0$. Hence we input the bounding polynomials

```
pmin = pol([0.45 1.95 2.95 5.95 3.95 3.95 1],6)

ppmin =
```

```

0.45 + 2s + 3s^2 + 6s^3 + 4s^4 + 4s^5 + s^6

pmax = pol([0.55 2.05 3.05 6.05 4.05 4.05 1],6)

pmax =

0.55 + 2s + 3s^2 + 6s^3 + 4s^4 + 4s^5 + s^6

```

and plot the Kharitonov rectangles for the critical range $0 \leq w \leq 1$ where they make the closest approach to $z = 0$:

```
khplot(pmin,pmax,0:.001:1)
```

In spection of Fig. 4 shows that $0 \notin p(jw, Q)$. Hence, the Zero Exclusion Condition is satisfied and we conclude that the interval polynomial is robustly stable.

Algorithm

The routine uses standard routines from the Polynomial Toolbox.

Diagnostics

The macro returns error messages in the following situations:

- The input arguments are not polynomial objects
- The input matrices have inconsistent dimensions
- There is an invalid input string argument

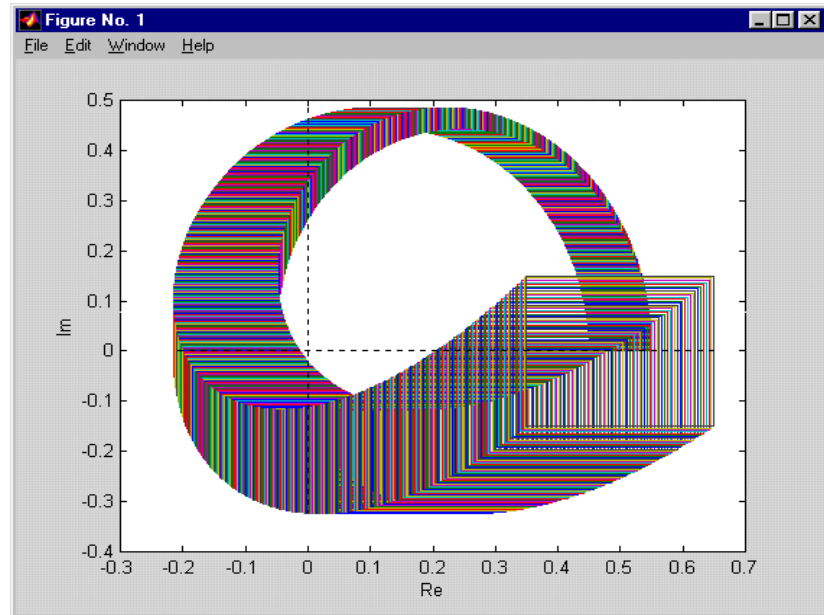


Fig. 4. Kharitonov rectangles for Example 2

The routine displays warning messages if

- The input polynomials are not of a continuous-time nature, that is, their variable is neither s nor p

- The interval polynomial is not degree invariant
- The input frequency is not real

See also

`kharit` Kharitonov polynomials

`ptopplot` plot polytopic value set

kron

Purpose Kronecker tensor product of polynomial matrices

Syntax `Z = kron(X,Y)`
`Z = kron(X,Y,tol)`

Description The command

`Z = kron(X,Y)`

forms the Kronecker tensor product of X and Y . The result is a large matrix formed by taking all possible products of the elements of X and those of Y . For instance, if X is 2×3 then `kron(X,Y)` is

$$\begin{bmatrix} X(1,1)*Y & X(1,2)*Y & X(1,3)*Y \\ X(2,1)*Y & X(2,2)*Y & X(2,3)*Y \end{bmatrix}$$

The command

`Z = kron(X,Y,tol)`

works with zeroing specified by the optional relative tolerance `tol`.

Example Define

`X = [1+s`

```

      s^2  ];

Y = [ 2  3-s
      0   1  ];

```

Then

```
Z = kron(X,Y)
```

returns

```

Z =

      2 + 2s      3 + 2s - s^2
      0           1 + s
      2s^2        3s^2 - s^3
      0           s^2

```

Algorithm

The routine uses standard routines from the Polynomial Toolbox.

Diagnostics

The macro displays a warning message if the input matrices have inconsistent variables.

See also

times array multiplication for polynomial matrix objects
mtimes matrix multiplication for polynomial matrix objects

lcoef

Purpose Various leading coefficient matrices of a polynomial matrix

Syntax

`lcoef(A)`

`lcoef(A, 'mat')`

`lcoef(A, 'ent')`

`lcoef(A, 'row')`

`lcoef(A, 'col')`

`lcoef(A, 'dia')`

Description

For a polynomial matrix A , the following leading coefficients are returned:

- `lcoef(A)` and `lcoef(A, 'mat')`: the leading (highest power) matrix coefficient
- `lcoef(A, 'ent')`: the matrix of scalar leading coefficients of the individual elements
- `lcoef(A, 'row')`: the matrix of row-leading coefficients
- `lcoef(A, 'col')`: the matrix of column-leading coefficients

If A is para-Hermitian ($A' = A$) then

- `lcoef(A,'dia')` returns its diagonal leading coefficient matrix.

Any of these syntaxes may be called with two output arguments. The second argument contains the vector or matrix of the corresponding degrees (which is the same as the first output argument of `deg`).

Examples

The simple polynomial matrix

```
P = [ 1  s
      s^2 0 ];
```

has the leading coefficient matrix and degree

```
[L,degP] = lcoef(P)
```

```
L =
    0    0
    1    0

degP =
    2
```

The degrees of the entries and corresponding leading coefficients are

```
[L,DEGP] = lcoef(P,'ent')
L =
```

```

1      1
1      0
DEGP =
0      1
2  -Inf

```

The row degrees and the row leading coefficient matrix follow as

```

[L, DegP] = lcoef(P, 'row')

L =
0      1
1      0

DegP =
1
2

```

The column degrees and column leading coefficient matrix are

```

[L, DegP] = lcoef(P, 'col')

L =

```

```

0      1
1      0
DegP =
2      1

```

Finally,

```

[L,DegZ] = lcoef([ 1+2*s^2    3*s
                  -3*s      4 ],'dia')

```

results in

```

L =
-2    -3
-3     4
DegZ =
1
0

```

Algorithm

The macro `deg` uses standard MATLAB operations.

Diagnostics

The macro displays an error message if

- an unknown option is encountered
- in case the option is '`dia`': if the input matrix is not square or not para-Hermitian

See also

`deg` various degree matrices of a polynomial matrix

ldiv, rdiv

Purpose Left or right polynomial matrix division

Syntax $[Q,R] = \text{ldiv}(N,D[,tol])$
 $[Q,R] = \text{rdiv}(N,D[,tol])$

Description Given a square polynomial matrix D and a polynomial matrix N with the same number of rows the command

$$[Q,R] = \text{ldiv}(N,D)$$

computes the polynomial matrix quotient Q and the polynomial matrix remainder R such that

$$N = DQ + R$$

and the degree of R is strictly less than the degree of D . Moreover, if D is row reduced then the degree of each row of R is strictly less than the degree of the corresponding row of D . If D is nonsingular and row reduced then the rational matrix $D^{-1}R$ is strictly proper and the matrices Q and R are unique.

There may be no solution if D is singular. In this case all the entries of Q and R are set equal to **NaN**.

A tolerance **tol** may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Given a square polynomial matrix D and a polynomial matrix N with the same number of columns the command

`[Q,R] = rdiv(N,D)`

computes the polynomial matrix quotient Q and the polynomial matrix remainder R such that

$$N = QD + R$$

and the degree of R is strictly less than the degree of D . Moreover, if D is column reduced then the degree of each column of R is strictly less than the degree of the corresponding column of D . If D is nonsingular and column reduced then the rational matrix RD^{-1} is strictly proper and the matrices Q and R are unique.

Examples

The command

`[Q,R] = ldiv(1+2*s+s^2,1+s)`

correctly returns

$Q =$

$1 + s$

Zero polynomial matrix: 1-by-1, degree: -Inf

$R =$

0

If N and D are defined as

```
D = [ 1+s 0
      -s^3-s^4 1+s ];
N = [ 0 1 ];
```

then

```
[Q,R] = rdiv(N,D)
```

returns

```
Q =
      1 - s + s^2 0
R =
     -1 - s^3 1
```

Algorithm

The macro `ldiv` is dual to `rdiv`. The algorithm in `rdiv` is as follows. Solve the polynomial matrix equation

$$\begin{bmatrix} R & Q \end{bmatrix} \begin{bmatrix} I \\ D \end{bmatrix} = N$$

for polynomial matrices R and Q such that the column degrees of R are strictly less than the corresponding column degrees of D . This is done with a proper column

selection scheme in the Sylvester resultant algorithm for solving (1). The linear system resulting from the Sylvester resultant algorithm is solved with QR factorization or SVD decomposition.

D is assumed to be column reduced. If it is not then D is reduced with `colred`. The correct remainder matrix R then is recovered with `xab`.

This algorithm is described by Zhang and Chen (1983).

Diagnostics

The macro issues an error message if

the second input polynomial matrix is missing

- invalid `Not-a-Number` or `Inf` entries are found in the input matrices
- an invalid third input argument is encountered

the input polynomial matrices have inconsistent dimensions

- empty polynomial matrices are encountered

A warning message follows in case of inconsistent variables for the input matrices or if division by zero occurs.

See also

<code>longldiv</code> , <code>longrdiv</code>	long left or right division of polynomial matrices
<code>colred</code> , <code>rowred</code>	column or row reduction of a polynomial matrix

ldivide (.\), rdivide (./)

Purpose Left or right array division

Syntax $X = A.\backslash B$

$X = B./A$

Description The command

$X = A.\backslash B$

denotes element-by-element division from the left. The matrices A and B must have the same dimensions unless A is a scalar. The entries of X are the solutions of the equations $A(i, j) * X(i, j) = B(i, j)$ as computed with the help of the function **axb**.

The command

$X = B./A$

denotes element-by-element division from the right. The matrices A and B must have the same dimensions unless A is a scalar. The entries of X are the solutions of the equations $X(i, j) * A(i, j) = B(i, j)$ as computed with the help of the function **xab**.

The functions **axb** and **xab** are called with their default tolerances.

Example

Define

```
A = 1+s;
B = [ 2+2*s  1+3*s+2*s^2 ];
```

Then

```
A.\B
```

returns

```
ans =
      2      1 + 2s
```

On the other hand,

```
B./(s+2)
```

returns

```
Constant polynomial matrix: 1-by-1
ans =
      NaN      NaN
```

This means that $s + 2$ does not divide A .**Algorithm**The routines call `axb` or `xab`.

Diagnostics

The macro displays a warning message if

- the input matrices have inconsistent variables
- invalid **Not-a-Number** or **Infinite** entries are encountered in the input matrices

In case of division by zero a warning follows.

See also

axb, xab solution of linear polynomial equations

mldivide, mrdivide polynomial matrix division

length, size

Purpose Length or size of a polynomial matrix

Syntax

```
l = length(P)

mn = size(P)

[m,n] = size(P)

[m,n,deg] = size(P)

m = size(P,1)

n = size(P,2)

deg = size(P,3)
```

Description The command

```
l = length(P)
```

returns the length of polynomial matrix P . It is equivalent to `max(size(P))` for a nonempty polynomial matrix and 0 for an empty polynomial matrix.

Given a polynomial matrix P , the command

```
mn = size(P)
```

returns the row vector `mn = [m n]` containing the number of rows m and columns n of the matrix. The command

```
[m,n,deg] = size(P)
```

returns the numbers of rows and columns as separate output variables.

The command

```
[m,n,deg] = size(P)
```

returns the number of rows m , the number of columns n , and the degree `deg` of the polynomial matrix P . The command

```
m = size(P,1)
```

returns just the number of rows,

```
n = size(P,2)
```

the number of columns, and

```
deg = size(P,3)
```

the degree.

Examples

Let

```
P = [ 1+s 2*s^2 3+4*s
      5      6      7      ];
```


Then

```
length(P)
```

results in

```
ans =
```

```
3
```

while

```
[n,m,deg] = size(P)
```

yields

```
n =
```

```
2
```

```
m =
```

```
3
```

```
deg =
```

```
2
```

Algorithm

The macros rely on standard MATLAB 5 and Polynomial Toolbox operations.

Diagnostics

The macro **length** display an error message if it has too many input arguments, and **size** protests if the second input argument is not 1, 2, or 3.

linvt, scale, shift

Purpose	<p>Linear transformation of the variable of a polynomial matrix</p> <p>Scale a polynomial matrix</p> <p>Shift the power of a polynomial matrix</p>
Syntax	<pre>Q = linvt(P,a,b) Q = linvt(P,a) Q = scale(P,a) [Q,a] = scale(P) Q = shift(P,n)</pre>
Description	<p>Linvt</p> <p>If P is a polynomial matrix and a and b are real numbers then</p> $Q = \text{linvt}(P,a,b)$ <p>computes the polynomial matrix Q such that</p> $Q(s) = P(as + b)$

If the third input variable b is missing then it is set equal to 0. The command also works for other variables than s .

Scale

Given a polynomial matrix P of degree n and a scalar a the function call

$$Q = \text{scale}(P, a)$$

results in the polynomial matrix Q given by

$$Q(s) = a^n P(s/a)$$

This leaves the leading coefficient matrix unaltered.

Without the second input argument, the call

$$[Q, a] = \text{scale}(P)$$

scales P automatically while taking the scaling factor as

$$a = \left(\frac{\|P_h\|}{\|P_l\|} \right)^{1/d}$$

P_l is the coefficient matrix of the term with the lowest degree, P_h the coefficient matrix of the term with the highest degree, and d the difference of the highest and lowest degrees.

Automatic scaling makes the norms of the coefficient matrix of the term with the lowest power equal to that of the highest power. Scaling a polynomial matrix sometimes improves the numerical accuracy of computations that involve the matrix.

Shift

The command

```
Q = shift(P,n)
```

computes

$$Q(s) = P(s)s^n$$

This also works for other variables than s .

Examples

Linvt

Let

```
P = [ 1+s  s^2  
      3*s  4  ];
```

Then we may substitute $s \mapsto s+1$ by the command

```
Q = linvt(P,1,-1)
```

This yields

$$Q = \frac{s^3 - 2s^2 + 1}{-3 + 3s}$$

Scale

Consider

$$P = -6000 + 1100s - 60s^2 + s^3;$$

We first scale by a factor 1/10:

$$Q = \text{scale}(P, 1/10)$$

MATLAB returns

$$Q = -6 + 11s - 6s^2 + s^3$$

The coefficients now all have the same order of magnitude. Automatic scaling results in

$$[R, a] = \text{scale}(P)$$

$$R = -1 + 3.3s - 3.3s^2 + s^3$$

```
a =
0.0550
```

The original polynomial may of course be retrieved as

```
scale(R,1/a)
ans =
-6e+003 + 1.1e+003s - 60s^2 + s^3
```

Shift

Again consider

```
P = [ 1+s  s^2
      3*s  4  ];
```

Then

```
Q = shift(P,2)
```

yields

```
Q =
s^2 + s^3      s^4
3s^3           4s^2
```

while

```
shift(Q,-1)
```

results in

```
ans =
```

$$\begin{array}{cc} s + s^2 & s^3 \\ 3s^2 & 4s \end{array}$$

Algorithm

The macros `linvt`, `scale` and `shift` use standard MATLAB and Polynomial Toolbox commands.

Diagnostics

The macro `linvt` complains if the second or third input argument is not a scalar.

The macro `scale` shows no error or warning messages.

The macro `shift` displays an error message if n is not an integer, and if the operation does not result in a polynomial matrix.

See also

`reverse` reverse the variable of a matrix fraction

llm, lrm

Purpose Least left or right multiple

Syntax $L = \text{llm}(N1, N2, \dots, Nk[, tol])$

$R = \text{lrm}(N1, N2, \dots, Nk[, tol])$

Description The command

$L = \text{llm}(N1, N2, \dots, Nk)$

computes a least left common multiple L of several polynomial matrices $N1, N2, \dots, Nk$ ($k > 0$) that all have the same number of columns. The multiple is defined as the least matrix divisible from the right by each of the matrices Ni . The matrices Mi such that $L = Mi * Ni$ may be recovered with the help of the command $Mi = L / Ni$, or by $Mi = \text{xab}(Ni, L)$.

The command

$R = \text{lrm}(N1, N2, \dots, Nk)$

computes a least right common multiple R of several polynomial matrices $N1, N2, \dots, Nk$ ($k > 0$) that all have the same number of rows. The multiple is defined as the least matrix divisible from the left by each of the Ni . The matrices Mi such that $R = Ni * Mi$ may be recovered with the help of the command $Mi = Ni \backslash R$, or by $Mi = \text{axb}(Ni, R)$.

In both cases a tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

A least common left multiple of

```
P = [ 1-s  s
      s  1+s ];
Q = [ 1  s
      s  1  ];
```

follows as

```
L = llm(P,Q)
L =
    -0.5 + 0.5s      -0.5 - 0.5s + s^2
    -0.5 + 0.5s + s^2    0.5 + 0.5s
```

Sure enough, there exist matrices

```
L1 = L/P, L2 = L/Q
L1 =
    -0.5 + 0.5s      -0.5 + 0.5s
```

```

      -0.5 - 0.5s      0.5 + 0.5s
L2 =
      -0.5 + s      -0.5
      -0.5      0.5 + s

```

such that

```
[L-L1*P L-L2*Q]
```

equals

```
Zero polynomial matrix: 2-by-4, degree: -Inf
```

```
ans =
```

```

      0      0      0      0
      0      0      0      0

```

Algorithm

The macro `llm` is the dual of `lrm`. The algorithm of `lrm` is based on the following idea. Suppose that we are to extract the least right multiple of a set of polynomial matrices $P_1(s)$, $P_2(s)$, ..., $P_n(s)$. Build the full row rank polynomial matrix

$$P(s) = \begin{bmatrix} P_1(s) & -P_2(s) & 0 & \cdots & \cdots & 0 \\ 0 & -P_2(s) & P_3(s) & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & (-1)^{n-1}P_{n-2}(s) & (-1)^n P_{n-1}(s) & 0 \\ 0 & \cdots & \cdots & 0 & (-1)^n P_{n-1}(s) & (-1)^{n+1} P_n(s) \end{bmatrix}$$

Let

$$K(s) = \begin{bmatrix} K_1(s) \\ K_2(s) \\ \cdots \\ K_n(s) \end{bmatrix}$$

be the polynomial matrix generating the minimal polynomial basis for the right null space of $P(s)$, that is, such that $P(s)K(s) = 0$. Then $R(s) = P_1(s)K_1(s)$ is the least right multiple of the polynomial matrices $P_1(s)$, $P_2(s)$, ..., $P_n(s)$. To see this, first note that R is a common multiple of the $P_i(s)$ since

$$R(s) = P_1(s)K_1(s) = P_2(s)K_2(s) = \cdots = P_n(s)K_n(s)$$

To prove the minimality of R consider another common multiple \bar{R} of the polynomial matrices P_i , and denote by \bar{K}_i the polynomial matrices such that

$$\bar{R}(s) = P_1(s)\bar{K}_1(s) = P_2(s)\bar{K}_2(s) = \cdots = P_n(s)\bar{K}_n(s)$$

Clearly, the polynomial matrix

$$\bar{K}(s) = \begin{bmatrix} \bar{K}_1(s) \\ \bar{K}_2(s) \\ \dots \\ \bar{K}_n(s) \end{bmatrix}$$

belongs to the polynomial right null space of P . Since K is a minimal basis for this null space, we have $K(s) = K(s)q(s)$ and $R(s) = R(s)q(s)$ for some polynomial matrix q . Hence, R is minimal.

Thus, the computation of a least right multiple amounts to extracting a minimal degree null space of a suitable polynomial matrix.

Diagnostics

The macro issues an error message under the following conditions.

- An invalid input argument is encountered
- Invalid **Not-a-Number** or **Inf** entries are found in the input matrices
- The input matrices have inconsistent dimensions

See also

gld, **grd** greatest divisors

lmf2dss, rmf2dss

Purpose Conversion of left or right matrix fraction representation to descriptor representation

Syntax

```
[a,b,c,d,e] = lmf2dss(N,D)
[a,b,c,d,e] = lmf2dss(N,D,tol)
[a,b,c,d,e] = rmf2dss(N,D)
[a,b,c,d,e] = rmf2dss(N,D,tol)
```

Description Consider two polynomial matrices $N(s)$ and $D(s)$ such that D is row reduced, which represent the system with transfer matrix

$$H(s) = D^{-1}(s)N(s),$$

The commands

```
[a,b,c,d,e] = lmf2dss(N,D)
[a,b,c,d,e] = lmf2dss(N,D,tol)
```

return a descriptor realization

$$\begin{aligned} \dot{e}x &= ax + bu \\ y &= cx + du \end{aligned}$$

of the system. If D and N are left coprime then the realization is minimal.

If D and N are polynomial matrices in the indeterminate variable p then the output arguments also define a continuous-time descriptor system.

If D and N are polynomial matrices in the indeterminate variables z , q , z^{-1} or d then the output arguments defines the discrete-time descriptor system

$$ex(t+1) = ax(t) + bu(t)$$

$$y(t) = cx(t) + du(t)$$

If D is column reduced then the commands

```
[a,b,c,d,e] = rmf2dss(N,D)
```

```
[a,b,c,d,e] = rmf2dss(N,D,tol)
```

return a descriptor realization of the right matrix fraction

$$H = ND^{-1}$$

The realization is minimal if N and D are right coprime.

The optional parameter `tol` is a tolerance. Its default value is the global zeroing tolerance.

Examples

Consider the SISO system with nonproper transfer function

$$H(s) = \frac{1 + 2s + s^2}{2 + s}$$

The commands

```
N = 1+2*s+s^2; D = 2+s;
[a,b,c,d,e] = lmf2dss(N,D)
```

return the descriptor realization

```
a =
    -2     0     0
     0     1     0
     0     0     1

b =
     1
     0
     1

c =
     1    -1     0

d =
     0
```



```
e =
    1     0     0
    0     0     1
    0     0     0
```

The commands

```
symbol(N,'z^-1'); N
symbol(D,'z^-1'); D
[a,b,c,d,e] = lmf2dss(N,D)
```

result in

```
N =
    1 + 2z^-1 + z^-2

D =
    2 + z^-1

a =
   -0.5000    1.0000
         0         0
```

```

b =
    1.5000
    1.0000

c =
    0.5000    0

d =
    0.5000

e =
    1    0
    0    1

```

Algorithm

For the conversion of the left fraction $H = D^{-1}N$ to state space form first **lmf2ss** is called to convert the fraction to generalized state space form which then is converted to descriptor form with the help of **ss2dss**. If N and D are polynomial matrices in z^{-1} or d then in **lmf2ss** the fraction is first converted to a fraction of polynomial matrices in z with the help of the macro **reverse**.

The macro **rmf2ss** works similarly.

Diagnostics

The macros **lmf2dss** and **rmf2dss** issue error messages if

- An invalid value for the tolerance is encountered

- The number of input arguments is incorrect
- The input matrices are not polynomial matrices
- The input matrices have inconsistent variables
- The input matrices have incompatible sizes

See also

<code>dss2lmf</code> , <code>dss2rmf</code>	conversion from descriptor representation to a left or right polynomial matrix fraction representation
<code>dss</code>	create a descriptor state space object or convert to a descriptor state space object
<code>lmf2ss</code> , <code>rmf2ss</code> <code>ss2lmf</code> , <code>ss2rmf</code>	conversion of a left or right matrix fraction to state space representation and vice-versa

lmf2rat, rmf2rat, lmf2tf, rmf2tf

Purpose Conversion from left or right matrix fraction to rational or transfer function format

Syntax `[num,den] = lmf2rat(N,D[,tol])`

`[num,den] = rmf2rat(N,D[,tol])`

`[num,den] = lmf2tf(N,D[,tol])`

`[num,den] = rmf2tf(N,D[,tol])`

Description Given two polynomial matrices N and D , where D is square and has the same number of rows as N , the function

`[num,den] = lmf2rat(N,D)`

returns the transfer function

$$H = D^{-1}N$$

in “rational” format. The polynomial matrices **num** and **den** have the same sizes. The (i, j) th entry of **num** contains the numerator of the corresponding entry H_{ij} of the transfer matrix H , and the corresponding entry of **den** contains its denominator.

The command

`[num,den] = lmf2tf(N,D)`

returns the transfer function H in Control System Toolbox format. Both **num** and **den** are cell arrays of the same size as the transfer matrix H . The (i, j) th cell of **num** contains the numerator polynomial of the transfer function and the corresponding cell of **den** the denominator polynomial. The polynomials are in MATLAB format. In the SISO case **num** and **den** are polynomials in MATLAB format. Note that the polynomials contained in **num** and **den** should be taken as polynomials in the variables of N and D .

Similarly, if N and D are two polynomial matrices such that D is square and has the same number of columns as N then the functions

```
[num,den] = rmf2rat(N,D)
```

```
[num,den] = rmf2tf(N,D)
```

return the transfer function

$$H = ND^{-1}$$

in rational and in Control System Toolbox format, respectively.

In all cases a tolerance **tol** may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Transfer function models are not recommended. State space and matrix fraction models are more natural and the conversion from and to transfer function models is numerically delicate.

Examples

Consider the 2×1 transfer function defined by the left matrix fraction $H = D^{-1}N$, with

```
D = [ 2+s  0
      1    1 ];
N = [ 1
      1 ];
```

The transfer matrix follows in rational form by typing

```
[num,den] = lmf2rat(N,D)
```

MATLAB returns

```
num =
      1
    1 + s
den =
      2 + s
      2 + s
```

Inspection shows that

$$H(s) = D^{-1}(s)N(s) = \left[\begin{array}{c} 1 \\ \frac{2+s}{1+s} \\ \frac{2+s}{2+s} \end{array} \right]$$

Algorithm

To convert the right fraction $H = ND^{-1}$ first the matrix D is inverted by computing its adjoint and determinant (see `adj`). After multiplying N on the right by the adjoint of D each of the rational entries of H is simplified by first using `gcd` to compute the greatest common divisor of the numerator and denominator and then canceling the divisor using `xab`.

The macros `lmf2rat` and `lmf2tf` are the duals of `rmf2rat` and `rmf2tf`.

Diagnostics

The macros return error messages if

- an invalid tolerance is encountered
- the number of input arguments is incorrect
- invalid `NaN` or `Inf` entries are found in the input polynomial matrices
- the input matrices have incompatible sizes
- the input matrices are not of the proper type
- the input matrices have different variable strings
- the factorization fails (that is, a common divisor cannot be canceled)

In the latter case it is recommended to modify the tolerance.

See also

`rat2lmf`, `rat2rmf`,
`tf2lmf`, `tf2rmf`

conversion from rational or Control System Toolbox transfer function format to a left or right polynomial matrix fraction

`tf`

create a Control System Toolbox LTI object in transfer function format

lmf2rmf, rmf2lmf

Purpose Polynomial matrix fraction conversions

Syntax

```
[P,Q] = lmf2rmf(N,D)
[P,Q] = lmf2rmf(N,D,tol)
[P,Q] = rmf2lmf(N,D)
[P,Q] = rmf2lmf(N,D,tol)
```

Description Given a nonsingular square polynomial matrix D and a polynomial matrix N with the same numbers of rows, the command

```
[P,Q] = lmf2rmf(N,D)
```

computes a nonsingular square polynomial matrix Q and a polynomial matrix P such that

$$D^{-1}N = PQ^{-1}$$

with P and Q right coprime. Hence, the routine converts a left polynomial matrix fraction into a coprime right one. If $D^{-1}N$ is proper then Q is column reduced.

Similarly, given a nonsingular square polynomial matrix D and a polynomial matrix N with the same numbers of columns, the command

`[P,Q] = rmf2lmf(N,D)`

computes a nonsingular square polynomial matrix Q and a polynomial matrix P such that

$$ND^{-1} = Q^{-1}P$$

with P and Q left coprime. Hence, the routine converts a right polynomial matrix fraction into a coprime left one. If ND^{-1} is proper then Q is row reduced.

For both routines a tolerance `tol` may be specified as an additional input argument. Its default value is the global zering tolerance.

Examples

To transform the left polynomial matrix fraction $D^{-1}N$, with

```
D = 1+2*s+s^2;
```

```
N = [1+s 3+4*s^2];
```

into a coprime right fraction, just type

```
[P,Q] = lmf2rmf(N,D)
```

MATLAB responds with

```
P =
```

```
1      -4 + 4s
```

$$Q = \begin{bmatrix} 1 + s & -7 \\ 0 & 1 + s \end{bmatrix}$$

As another application consider the conversion of the rational matrix

$$R(s) = \begin{bmatrix} \frac{1}{2+s} \\ \frac{1}{2+s} \end{bmatrix}$$

to a coprime left matrix fraction. Inspection shows that

$$R(s) = \begin{bmatrix} 2+s & 0 \\ 0 & 2+s \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1+s \end{bmatrix}$$

but this fraction is not coprime. We therefore write

$$R(s) = N(s)D^{-1}(s)$$

with

$$\begin{aligned} N &= \begin{bmatrix} 1 \\ 1+s \end{bmatrix}; \\ D &= 2+s; \end{aligned}$$

and apply

```
[P,Q] = rmf2lmf(N,D)
```

This results in the left coprime fraction $R = Q^{-1}P$ with

Constant polynomial matrix: 2-by-1

P =

1

1

Q =

$2 + s$ 0

1 1

Algorithm

The macro **lmf2rmf** uses the routine **null** to compute $[P; Q]$ so that its columns form a basis for the right null space of $[D - M]$. The macro **rmf2lmf** is the dual of **lmf2rmf**.

Diagnostics

The macros **lmf2rmf** and **rmf2lmf** issue error messages if

- The second input argument is missing
- One or both of the input matrices are empty
- The input matrices have inconsistent dimensions

See also

<code>null</code>	null space of a polynomial matrix
<code>minbasis</code>	minimal polynomial basis

lmf2ss, rmf2ss

Purpose	Conversion of left or right matrix fraction representation to state space representation
Syntax	$[a,b,c,d] = \text{lmf2ss}(N,D)$ $[a,b,c,d] = \text{lmf2ss}(N,D,\text{tol})$ $[a,b,c,d] = \text{rmf2ss}(N,D)$ $[a,b,c,d] = \text{rmf2ss}(N,D,\text{tol})$
Description	<p>Given two polynomial matrices $N(s)$ and $D(s)$ such that D is nonsingular and row reduced the commands</p> $[a,b,c,d] = \text{lmf2ss}(N,D)$ $[a,b,c,d] = \text{lmf2ss}(N,D,\text{tol})$ <p>return the (generalized) observer-form realization</p> $\dot{x} = ax + bu$ $y = cx + d(s)u$ <p>with s the differentiation operator, of the system with transfer matrix</p> $H(s) = D^{-1}(s)N(s)$

If D and N are polynomial matrices in the indeterminate variable p then the output arguments also define a continuous-time state space system, and d is returned as a polynomial matrix in p .

If D and N are polynomial matrices in the indeterminate variables z , q , z^{-1} or d then the output arguments define the discrete-time system

$$\begin{aligned}x(t+1) &= ax(t) + bu(t) \\ y(t) &= cx(t) + d(z)u(t)\end{aligned}$$

where z is the time shift operator given by $zu(t) = z(t+1)$. The polynomial matrix d is returned as a polynomial matrix in z if N and D are polynomial matrices in z or z^{-1} , and as a polynomial matrix in q if N and D are polynomial matrices in q or d .

If d is a constant matrix then it is always returned in MATLAB format.

If D is nonsingular and column reduced then the commands

$$\begin{aligned}[\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}] &= \text{rmf2ss}(\mathbf{N}, \mathbf{D}) \\ [\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}] &= \text{rmf2ss}(\mathbf{N}, \mathbf{D}, \text{tol})\end{aligned}$$

return a (generalized) controller-form realization of the right matrix fraction

$$H = ND^{-1}$$

The optional parameter `tol` is a tolerance that is used in testing whether D is row or column reduced. Its default value is the global zeroing tolerance.

Example

Consider the SISO system with nonproper transfer function

$$H(s) = \frac{1 + 2s + s^2}{2 + s}$$

The commands

```
N = 1+2*s+s^2; D = 2+s;
```

```
[a,b,c,d] = lmf2ss(N,D)
```

return the state space realization

```
a =
```

```
    -2
```

```
b =
```

```
    1
```

```
c =
```

```
    1
```

```
d =
```

```
    s
```


The commands

```
N.v = 'z^-1', D.v = 'z^-1'
```

```
[a,b,c,d] = lmf2ss(N,D)
```

result in

```
N =
```

```
1 + 2z^-1 + z^-2
```

```
D =
```

```
2 + z^-1
```

```
a =
```

```
-0.5000    1.0000
```

```
0          0
```

```
b =
```

```
1.5000
```

```
1.0000
```

```
c =
```

```
0.5000    0
```

$d =$
0.5000

Algorithm

The macro **lmf2ss** is the dual of **rmf2ss**.

For the conversion of the right fraction $H = ND^{-1}$ to state space form first the strictly proper and polynomial parts of H are separated by polynomial division as

$$H(s) = N_o(s)D^{-1}(s) + d(s)$$

The controller-form realization of the strictly proper part $H(s) = N_o(s)D^{-1}(s)$ is constructed as described in Kailath (1980). The column degrees of the column reduced polynomial matrix D are the controllability indices of the system.

If N and D are polynomial matrices in the indeterminate variable z^{-1} or d then the fraction is first converted to a fraction in z using the routine **reverse**.

Diagnostics

The macros **lmf2ss** and **rmf2ss** issue error messages under the following conditions.

- The second input argument is missing
- An invalid **NaN** or **Inf** entry is encountered in the input polynomial matrices
- The input matrices have inconsistent dimensions
- The third input argument is not a scalar
- Inconsistent variables are encountered in the input

- The second input matrix is singular
- The second input matrix is not column or row reduced as required

See also

<code>ss2lmf, ss2rmf</code>	conversion from state space representation to a left or right polynomial matrix fraction representation
<code>lmf2dss, rmf2dss, dss2lmf, dss2rmf</code>	conversion of a left or right matrix fraction to descriptor representation and vice-versa
<code>reverse</code>	reverse the variable of a polynomial matrix fraction

lmf2zpk, rmf2zpk

Purpose Conversion of a left or right matrix fraction to zero-pole-gain (ZPK) format

Syntax `[Z,P,K] = lmf2zpk(N,D[,tol])`

`[Z,P,K] = rmf2zpk(N,D[,tol])`

Description Given two polynomial matrices N and D , where D is square and has the same number of rows as N , the function

$$[Z,P,K] = \text{lmf2zpk}(N,D)$$

returns the transfer function

$$H = D^{-1}N$$

in zero-pole-gain format. In this format, each entry

$$H_{ij}(s) = k \frac{\prod_j (s - z_j)}{\prod_j (s - p_j)}$$

of the transfer matrix is characterized by its zeros $z_j, j = 1, 2, \dots$, its poles $p_j, j = 1, 2, \dots$, and its gain k .

Z and P are cell arrays and K is a two-dimensional array. Each cell of Z contains the zeros of the corresponding transfer function, each cell of P contains the poles, and each entry of K contains the gain. Note that the polynomials whose roots and gains are represented are polynomials in the variables of N and D (that is, s or p in the continuous-time case, and z , q , d or z^{-1} in the discrete-time case).

Similarly, if N and D are two polynomial matrices such that D is square and has the same number of columns as N then the function

```
[Z,P,K] = rmf2zpk(N,D)
```

returns the transfer function

$$H = ND^{-1}$$

in zero-pole-gain format.

In both cases a tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Consider the system with transfer matrix $H = D^{-1}N$, where

```
D = [ 2+s  0
      1    1 ];
N = [ 1
      1 ];
```

The command

```
[Z,P,K] = lmf2zpk(N,D)
```

yields the ZPK data

```
Z =  
  
[]  
  
[-1.0000]  
  
P =  
  
[-2]  
  
[-2]  
  
K =  
  
1.0000  
  
1.0000
```

Inspection shows that the transfer matrix of the system is

$$H(s) = \begin{bmatrix} \frac{1}{s+2} \\ \frac{s+1}{s+2} \end{bmatrix}$$

If N and D are redefined as matrices in z^{-1} rather than s then the same ZPK data are returned, but the transfer matrix is to be interpreted as

$$H(z^{-1}) = \begin{bmatrix} \frac{1}{z^{-1} + 2} \\ \frac{z^{-1} + 1}{z^{-1} + 2} \end{bmatrix}$$

Algorithm

Using the routine `lmf2tf` the fraction $H = D^{-1}N$ is converted to transfer function form. The numerators and denominators of the entries of H are converted to ZPK format with the help of `pol2root`.

The macro `rmf2zpk` is the dual of `lmf2zpk`.

Diagnostics

The macros `lmf2zpk` and `rmfzpk` return error messages if there is no second input argument.

See also

<code>zpk2lmf</code>	conversion from transfer function format to a left or right
<code>zpk2rmf</code>	polynomial matrix fraction
<code>zpk</code>	create a Control System Toolbox LTI object in ZPK format

longldiv, longrdiv

Purpose Long left or right polynomial matrix division

Syntax `[Q,R] = longldiv(N,D,d[,tol])`

`Q = longldiv(N,D,d[,tol])`

`[Q,R] = longrdiv(N,D,d[,tol])`

`Q = longrdiv(N,D,d[,tol])`

Description Let D be a square polynomial matrix and N a polynomial matrix with the same numbers of rows. Then if both D and N are polynomial matrices in the “discrete-time” variable z , the command

`[Q,R] = longldiv(N,D,d)`

returns the polynomial matrices $Q(z)$ and $R(z^{-1})$ (the latter of degree d) that together define the first $n + d + 1$ terms of the Laurent series expansion

$$D^{-1}(z)N(z) = Q_n z^n + Q_{n-1} z^{n-1} + \dots + Q_1 z + R_0 + R_1 z^{-1} + R_2 z^{-2} + \dots + R_d z^{-1} + \dots$$

of the left fraction $D^{-1}N$ about the point $z = \infty$. The coefficient matrices $Q_n, Q_{n-1}, \dots, Q_1, R_0, R_1, \dots$, constitute the impulse response of the discrete-time system defined by the polynomial matrix fraction. If the fraction is nonproper then $n > 0$ and the system is non-causal.

Also if N and D are polynomial matrices in q , d or z^{-1} the correct series expansion is returned, with q being taken as synonymous with z and d to z^{-1} .

If N and D are polynomial matrices in the “continuous-time” variable s (or p , which is taken to be synonymous with s) then the command

[Q,R] = longldiv(N,D,d)

returns the first $n + d + 1$ terms of the Laurent series expansion

$$D^{-1}(s)N(s) = Q_n s^n + Q_{n-1} s^{n-1} + \dots + Q_1 s + R_0 + R_1 s^{-1} + R_2 s^{-2} + \dots + R_d s^{-d} + \dots$$

of $D^{-1}N$ about the point $s = \infty$. Q is a polynomial matrix in s and R is a three-dimensional array R such that

$$R_k = R(:, :, k+1), \quad k = 0, 1, 2, \dots, d$$

The coefficients R_k , $k = 1, 2, \dots$, are the Markov parameters of the system.

The command

Q = longldiv(N,D,d)

returns just the polynomial part of $D^{-1}N$, including the zero-degree term, that is,

$$Q(v) = R_0 + Q_1 v + Q_2 v^2 + \dots + Q_n v^n$$

The command

[Q,R] = longrdiv(N,D,d)

expands the right fraction ND^{-1} , provided N and D have the same number of columns.

Both for `longldiv` and `longrdiv` an optional tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Let

```
N = 1; D = z-0.5;
```

Then

```
[Q,R] = longldiv(N,D,5)
```

returns

```
Zero polynomial matrix: 1-by-1, degree: -Inf
```

```
Q =
```

```
0
```

```
R =
```

```
z^-1 + 1/2*z^-2 + 1/4*z^-3 + 1/8*z^-4 + 1/16*z^-5
```

In the format

```
gprop symbr, R
```

we see that

`R =`

$$z^{-1} + 1/2z^{-2} + 1/4z^{-3} + 1/8z^{-4} + 1/16z^{-5}$$

As another example consider

`N = 1+z;`

`D = 1+z +2*z^2;`

We then may compute and plot the impulse response of this system according to

`d = 20; [P,Q] = longldiv(N,D,d);`

`t =0:d; ir = Q{:};`

`plot(t,ir,t,ir,'ro')`

The result is shown in Fig. 5.

As a continuous-time example consider the left fraction defined by

`D = 1+s;`

`N= [1+s+s^2 1];`

`[Q,R] = longldiv(N,D,6);`

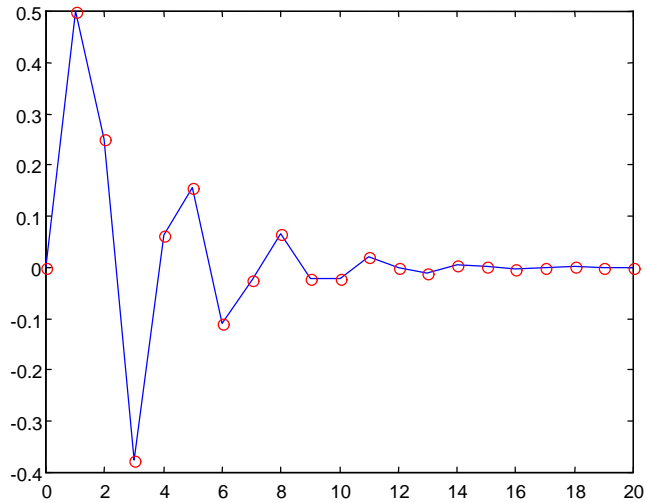


Fig. 5. Impulse response

The nonproper part of $D^{-1}N$ is

Q

$Q =$

$s \quad 0$

while the first 6 Markov parameters $R_k = R(:, :, k)$, $k = 1, 2, \dots, 6$, follow from

R

$R(:, :, 1) =$

0 0

$R(:, :, 2) =$

1 1

$R(:, :, 3) =$

-1 -1

$R(:, :, 4) =$

1 1

$R(:, :, 5) =$

-1 -1

$R(:, :, 6) =$

1 1

$R(:, :, 7) =$

-1 -1

Algorithm

1. If N and D are polynomial in the variables d or z^{-1} then the fraction is first converted to a fraction in z using **reverse**.
2. If D is not row-reduced in the case of **longldiv** or not column reduced in the case of **longrdiv** then it is first reduced.
3. The polynomial part Q is computed using **ldiv** or **rdiv**.
4. After multiplying the remainder of this division by z^d or s^d as the case may be the desired series expansion is found by dividing by D , again using **ldiv** or **rdiv**.

Diagnostics

The macro issues an error message under the following conditions:

- There are not enough input arguments
- The third input is not a nonnegative integer
- The denominator matrix is not square
- The input polynomial matrices have inconsistent dimensions

A warning follows if

- the denominator is singular to working precision
- the input matrices have inconsistent variables

See also

`ldiv`, `rdiv`

left and right polynomial matrix division

lti2lmf, lti2rmf

Purpose Conversion of an LTI object to a left or right matrix fraction representation

Syntax `[N,D] = lti2lmf(sys[,tol])`

`[N,D] = lti2rmf(sys[,tol])`

Description The command

`[N,D] = lti2lmf(sys)`

converts the LTI object `sys` in Control System Toolbox format to the left polynomial matrix fraction format

$$H = D^{-1}N$$

The LTI object may be a state space model, a descriptor state space model, a transfer function model, or a zero-pole-gain model.

If the object is a continuous-time system then D and N are returned as matrices in the variable s . If it is a discrete-time system then D and N are returned as matrices in the variable z .

The command

`[N,D] = lti2lmf(sys)`

returns the right fraction

$$H = ND^{-1}$$

In both cases a tolerance `tol` may be specified as an additional input argument.

The functions `lti2lmf` and `lti2rmf` only work if the Control System Toolbox is included in the MATLAB path.

Later versions of the Polynomial Toolbox will support a polynomial matrix fraction LTI object.

Examples

The Control System Toolbox command

```
sys = ss(1,2,3,4)
```

creates the LTI system in state space form

```
a =
      0      0
      0      0
      0      0
      0      0

b =
      0      0
      0      0
      0      0
      0      0

c =
      0      0
      0      0
      0      0
      0      0
```

```

                                x1
                                3
                                y1
                                4
                                u1
                                y1

```

Continuous-time system.

From this we obtain

```
[N,D] = lti2lmf(sys)
```

N =

```
0.67 + 1.3s
```

D =

```
-0.33 + 0.33s
```

The Control System Toolbox LTI object in transfer function form

```
sys = tf([1 2],[3 4],1)
```

Transfer function:

```
z + 2
```

```
-----
```

```
3 z + 4
```

```
Sampling time: 1
```

is correctly converted by typing

```
[N,D] = lti2lmf(sys)
```

```
N =
```

```
1.5 + 0.75z
```

```
D =
```

```
3 + 2.3z
```

Algorithm

Depending on the type of LTI object the system is transformed into polynomial matrix fraction form by one of the convertors **ss2lmf**, **ss2rmf**, **tf2lmf**, **tf2rmf**, **zpk2lmf** or **zpk2rmf**.

Diagnostics

The macros return error messages if if the first input argument is not an LTI object or an incorrect value of the tolerance is encountered.

See also

ss, **tf**, **zpk** create a Control System Toolbox LTI object in **ss**, **tf** or **zpk** format

lu

Purpose LU factorization for polynomial matrices

Syntax `[L,U] = lu(A[,tol])`

`[L,U,P] = lu(A[,tol])`

Description Given a square polynomial matrix A with nonsingular constant term the command

`[L,U] = lu(A)`

computes an upper triangular polynomial matrix U whose diagonal entries have nonzero real parts and a unimodular matrix L such that

$$A = LU$$

$L(0)$ is “psychologically lower triangular,” that is, it is a product of lower triangular and permutation matrices.

The command

`[L,U,P] = lu(A)`

returns a unimodular matrix L with lower triangular $L(0)$, an upper triangular matrix U and a permutation matrix P so that $PA = LU$.

A tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Consider

$$P = \begin{bmatrix} 1+s & s^2 \\ s & 5 \end{bmatrix};$$

The command

$$[L,U] = \text{lu}(P)$$

returns

$$L = \begin{bmatrix} 1 + s & -s \\ s & 1 - s \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & 5s + s^2 - s^3 \\ 0 & 5 + 5s - s^3 \end{bmatrix}$$

Algorithm

The macro first calls `tri` to transform P unimodularly to upper triangular form. Next, standard LU decomposition is applied to the constant term.

Diagnostics

The macro issues error messages under the following conditions:

- Invalid **Not-a-Number** or **Inf** entries are encountered in the input matrix

- The input matrix is not square
- The constant term of the input matrix is singular
- The reduction to triangular form fails

See also

`tri` transformation of a polynomial matrix to triangular form

mat2pol

Purpose Conversion of a polynomial or polynomial matrix in MATLAB or Control System Toolbox format to Polynomial Toolbox format

Syntax `P = mat2pol(M)`

Description The command

`P = mat2pol(M)`

converts the polynomial or polynomial matrix M in MATLAB or Control System Toolbox format to Polynomial Toolbox format.

In MATLAB format a polynomial is represented as a row vector whose entries are the coefficients of the polynomial according to descending powers. In Control System Toolbox format a polynomial matrix is a cell array of the same dimensions as the polynomial matrix, with each cell a row vector representing the corresponding entry of the polynomial matrix in MATLAB format. If the polynomial matrix consists of a single element then in Control System Toolbox format it is represented as a row vector with entries in MATLAB format.

To comply with an older standard in the Control System Toolbox, polynomial matrices consisting of a single column may be represented as a matrix whose rows are the polynomial entries in MATLAB format. The macro `mat2pol` automatically converts this older format to a polynomial matrix object.

Examples

Conversion of a polynomial in MATLAB format to Polynomial Toolbox format:

```
P = mat2pol([1 2 3])
```

```
P =
```

```
3 + 2s + s^2
```

Conversion of a polynomial matrix in Control System Toolbox format. We first display a matrix in Control System Toolbox format

```
M
```

```
M =
```

```
[1x3 double]    [1x3 double]
```

Its entries are

```
M{1,1}, M{1,2}
```

```
ans =
```

```
1      2      0
```

```
ans =
```

```
3      0      4
```

Next we convert to Polynomial Toolbox format:


```
P = mat2pol(M)
```

```
P =
```

```
      2s + s^2      4 + 3s^2
```

Finally,

```
M = [ 1 2
```

```
      3 4 ];
```

is also a polynomial matrix in the older Control System Toolbox format. In Polynomial Toolbox format we can view it more easily:

```
mat2pol(M)
```

```
ans =
```

```
      2 + s
```

```
      4 + 3s
```

Algorithm

The macro `mat2pol` uses standard MATLAB 5 operations.

Diagnostics

The macro `mat2pol` displays an error message if the input is not a polynomial in MATLAB format or a polynomial matrix in Control System Toolbox format.

See also

`pol2mat` conversion from Polynomial Toolbox format to MATLAB or Control System Toolbox format

Commands — Online reference

<code>pol2dsp</code>	conversion from Polynomial Toolbox format to DSP format
<code>dsp2pol</code>	conversion from DSP format to Polynomial Toolbox format

minbasis

Purpose Minimal polynomial basis

Syntax `[B,rk] = minbasis(A[,tol])`

Description The command

```
[B,rk] = minbasis(A)
```

computes a minimal polynomial basis B for the submodule spanned by the columns of the polynomial matrix A . The basis is minimal in the sense of Forney, that is, B is column reduced and has full column rank rk . Moreover, the columns of B are arranged according to decreasing degrees. If C is another polynomial basis then necessarily $\deg(C, 'col') \geq \deg(B, 'col')$.

A tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples Consider

```
A = [ 1+s      s^2      1+s^3      4
      2      3+4*s+s^2      5      6 ];
```

The command

```
[B,rk] = minbasis(A)
```

yields

Constant polynomial matrix: 2-by-2

B =

1 4

5 6

rk =

2

On the other hand,

```
minbasis([ 1  s  s^2
           s^2 s  1  ])
```

returns

ans =

s^2 1

1 1

Algorithm

The algorithm to compute the minimal basis involves the following steps.

1. Given the polynomial matrix A , extract a greatest common right divisor D using the macro **grd**.
2. Compute the polynomial matrix L such that $A = LD$ using the macro **xab**.
3. Reduce L to column-reduced form B using the macro **colred**.

A detailed description of what a minimal basis for a polynomial submodule is provided in Forney (1975).

Diagnostics

The macro issues error messages under the following conditions:

- Invalid **Not-a-Number** or **Inf** entries are encountered in the input matrix
- An invalid second input argument is found

See also

colred column reduction of a polynomial matrix
null null space of a polynomial matrix

minus, plus, uminus, uplus

Purpose Subtraction, addition, unary minus and unary plus of polynomial matrices

Syntax

$$C = A - B$$

$$C = \text{minus}(A, B[, \text{tol}])$$

$$C = A + B$$

$$C = \text{plus}(A, B[, \text{tol}])$$

$$B = -A$$

$$B = \text{uminus}(A)$$

$$B = +A$$

$$B = \text{uplus}(A)$$
Description

The commands

$$C = A - B$$

$$C = \text{minus}(A, B)$$

subtract the polynomial matrix **B** from **A** with zeroing using the global zeroing tolerance. The command

```
C = minus(A,B,tol)
```

works with zeroing specified by the relative tolerance `tol`.

The commands

```
C = A + B
```

```
C = plus(A,B[,tol])
```

work similarly but return the sum of the polynomial matrices `A` and `B`.

The function of the commands

```
B = -A
```

```
B = uminus(A)
```

```
B = +A
```

```
B = uplus(A)
```

is obvious.

Example

The commands

```
A = 1+s/3;
```

```
B = 1+0.333333*s;
```

```
C = A-B
```

result in

```
C =  
3.3e-007s
```

but

```
minus(A,B,1e-4)
```

returns

```
Zero polynomial matrix: 1-by-1, degree: -Inf  
ans =  
0
```

Algorithm

The routines use standard MATLAB 5 operations.

Diagnostics

The macros **minus** and **plus** display an error message if

- they do not have enough input arguments
- the dimensions of the polynomial matrices do not agree

If the input matrices have inconsistent variables then a warning follows.

See also

times, **mtimes** polynomial multiplication

mixeds

Purpose

Solution of a SISO mixed sensitivity problem

Syntax

```
[y,x,gopt] = mixeds(n,m,d,a1,b1,a2,b2,gmin,gmax,accuracy)
[y,x,gopt] = mixeds(n,m,d,a1,b1,a2,b2,gmin,gmax,accuracy,tol)
[y,x,gopt] =
mixeds(n,m,d,a1,b1,a2,b2,gmin,gmax,accuracy,'show')
[y,x,gopt] = ...
    mixeds(n,m,d,a1,b1,a2,b2,gmin,gmax,accuracy,tol,'show')
```

Description

The command

```
[y,x,gopt] = ...
    mixeds(n,m,d,a1,b1,a2,b2,gmin,gmax,accuracy,tol,'show')
```

computes a compensator $C = y/x$ that solves the mixed sensitivity problem for the SISO plant $P = n/d$ with weighting functions $V = m/d$, $W_1 = a1/b1$ and $W_2 = a2/b2$.

The mixed sensitivity problem consists of minimizing

$$\sup_{w \in \mathbb{R}} |V(jw)|^2 (|W_1(jw)S(jw)|^2 + |W_2(jw)U(jw)|^2)$$

where

$$S = \frac{1}{1+PC}, \quad U = \frac{C}{1+PC}$$

are the sensitivity and input sensitivity functions of the closed-loop system, respectively.

The input parameters n , d , m , **a1**, **b1**, **a2** and **b2** are scalar polynomials or constants. The polynomial m needs to have the same degree as d . The polynomials m , **b1** and **b2** need to be strictly Hurwitz.

The input parameters **gmin** and **gmax** are lower and upper bounds for the minimal value of the mixed sensitivity criterion, respectively. The parameter **accuracy** specifies how closely the minimal norm is to be approached.

The optional input parameter **tol** = [**tolcnc1** **tolstable** **tolspf** **tolrr**] defines four tolerances.

1. The tolerance **tolcnc1** is used in canceling identical pole-zero pairs in the transfer function $C = y/x$ of the optimal compensator. Its default value is **1e-4**.
2. The tolerance **tolstable** is used in the various stability tests. It has the default value **1e-8**.

3. The tolerance `tolspf` is used in the spectral factorization, and has the default value `1e-8`.
4. The tolerance `tol1r` is used in the left-to-right and right-to-left conversions. Its default value is `1e-8`.

If the optional input argument `'show'` is present then the successive test values `gamma` of the minimal value of the criterion during the binary search are shown, and any pole-zero pair that is canceled in y/x is reported.

The output parameter `gopt` is the computed minimal value of the criterion.

Example

Consider the mixed sensitivity problem for the plant with transfer function

$$P(s) = \frac{n(s)}{d(s)} = \frac{1}{s^2}$$

defined by

$$V(s) = \frac{m(s)}{d(s)} = \frac{1 + s\sqrt{2} + s^2}{s^2}, \quad W_1(s) = \frac{a_1(s)}{b_1(s)} = 1, \quad W_2(s) = \frac{a_2(s)}{b_2(s)} = c(1 + rs)$$

We let $c = 0.1$, $r = 0.1$. First define the input parameters:

```
c = 0.1; r = 0.1; n = 1; d = s^2; m = s^2+s*sqrt(2)+1;
a1 = 1; b1 = 1; a2 = c*(1+r*s); b2 = 1;
gmin = 1; gmax = 10; accuracy = 1e-4;
```

Next we call the routine `mixeds`:

```
[y,x,gopt] = ...
    mixeds(n,m,d,a1,b1,a2,b2,gmin,gmax,accuracy,'show')
```

This is the response of MATLAB:

```
gamma      test result
-----
10         stable
1          no solution
5.5        stable
3.25       stable
2.125      stable
1.5625     stable
1.28125    unstable
1.42188    stable
1.35156    unstable
1.38672    stable
```

```
1.36914    unstable
1.37793    unstable
1.38232    unstable
1.38452    stable
1.38342    stable
1.38287    unstable
1.38315    unstable
1.38329    unstable
1.38335    stable
Cancel root at -0.999535
y =
      57 + 96s
x =
      79 + 16s + s^2
gopt =
      1.3834
```

The minimal value of the criterion is 1.3834 and the optimal compensator has the transfer function

$$C(s) = \frac{y(s)}{x(s)} = \frac{57 + 96s}{79 + 16s + s^2}$$

It is easy to compute the closed-loop poles of the optimal system:

```
clpoles = roots(d*x+n*y)

clpoles =

    -7.3282 + 1.8769i
    -7.3282 - 1.8769i
    -0.7071 + 0.7071i
    -0.7071 - 0.7071i
```

Algorithm

The algorithm is described in Kwakernaak (1996). Given an upper bound g for the criterion a suboptimal compensator is found by a polynomial matrix spectral factorization followed by a left-to-right conversion of a polynomial matrix fraction, and another polynomial matrix spectral factorization. By testing whether a stabilizing suboptimal solution exists the optimal solution is approached by a binary search on g .

Numerical degeneracy near the optimal solution is avoided by using a modified form of the spectral factorization. The optimal compensator often has a coinciding or nearly coinciding pole-zero pair. If detected such a pair is cancelled.

Diagnostics

The macro **mixed**s displays error messages in the following situations:

- Unknown option
- Erroneous tolerance parameter
- The polynomials d and m have different degrees
- The polynomials m , **b1** or **b2** are not Hurwitz
- The lower bound **gmin** exceeds the upper bound **gmax**
- No solution exists at the upper bound **gmax**. In this case **gmax** should be increased
- No stabilizing solution exists at the upper bound **gmax**. In this case **gmax** should also be increased
- A stabilizing solution exists at the lower bound **gmin**. In this case **gmin** should be decreased

If the option '**show**' is present then the successive values of the upper bound of the norm during the binary search are shown, and any pole-zero pair that is cancelled in the optimal compensator $C = y/x$ is reported.

See also

<code>dsshinf</code>	H_∞ suboptimal compensator for descriptor systems
<code>dssrch</code>	search for H-infinity optimal compensator for descriptor systems
<code>plqg, splqg</code>	LQG optimal compensator
<code>spf</code>	polynomial spectral factorization
<code>lmf2rmf, rmf2lmf</code>	Left-to-right and right-to-left matrix fraction conversion

mldivide (\), mrdivide (/)

Purpose Backslash or left polynomial matrix division

Slash or right polynomial matrix division

Syntax

`X = A\B`

`X = mldivide(A,B)`

`X = B/A`

`X = mrdivide(A,B)`

Description

The commands

`X = A\B`

`X = mldivide(A,B)`

are equivalent to

`X = axb(A,B)`

They determine a polynomial matrix X such that $AX = B$.

The commands

`X = B/A`

```
X = mrdivide(A,B)
```

are the same as

```
X = xab(A,B)
```

They determine a polynomial matrix X such that $XA = B$.

Example

Define

```
A = [ 1+s  1
      0    1 ];
```

```
B = [ 2+2*s
      1+s ];
```

Then

```
X = A\B
```

yields

```
X =
      1
      1 + s
```

while

```
Y = B/s
```

results in

```
Constant polynomial matrix: 2-by-1
```

```
Y =
```

```
NaN
```

```
NaN
```

This means that s does not divide B on the right.

Algorithm

See the description of **axb** for the algorithm.

Diagnostics

The macros display an error message if invalid **Not-a-Number** or **Infinite** entries are encountered in the input matrices. A warning message is issued in case of division by zero.

See also

axb , xab	solution of linear polynomial matrix equations
ldivide , rdivide	left or right array division

mono

Purpose Create a power of the current global variable string

Syntax `P = mono(n)`

Description The command

`P = mono(n)`

creates the n th power of the current global variable string. If the current global variable string is s then P is returned as s^n .

If n is a vector or matrix then the vector or matrix of monomials is returned.

Examples Typing

`P = mono([0 1 2])`

returns

`P =`

`1 s s^2`

More complex polynomials can be created for instance as

`P = sum(mono(0:5))`

which yields

```
P =
1 + s + s^2 + s^3 + s^4 + s^5
```

Other examples are

```
P = (1:5)*mono(0:4)'
P =
1 - 2s + 3s^2 - 4s^3 + 5s^4
```

and

```
P = (1:5)'*mono(0:4)
P =
1      s      s^2      s^3      s^4
2      2s     2s^2     2s^3     2s^4
3      3s     3s^2     3s^3     3s^4
4      4s     4s^2     4s^3     4s^4
5      5s     5s^2     5s^3     5s^4
```

Algorithm	The macro uses standard MATLAB 5 commands.
Diagnostics	<p>The macro returns an error message under the following conditions:</p> <ul style="list-style-type: none">▪ There are not enough input arguments▪ The input argument is not a MATLAB matrix▪ A negative or nonintegral degree is encountered
See also	<code>d</code> , <code>s</code> , <code>p</code> , <code>q</code> , <code>v</code> , <code>z</code> , <code>zi</code> create simple basic polynomials

mpower (^), power (.^)

Purpose Matrix power of a polynomial matrix
 Element-wise power of a polynomial matrix

Syntax `X = A^n`
 `X = mpower(A,n[,tol])`
 `X = A.^n`
 `X = power(A,n[,tol])`

Description The commands

`An = A^n`

`An = mpower(A,n)`

return A to the power n if n is a scalar nonnegative integer and A is square. This works with zeroing activated through the global zeroing variable. The command

`An = mpower(A,n,tol)`

works with zeroing specified by the input tolerance `tol`.

The commands

```
Z = X.^Y
```

```
Z = power(X,Y)
```

denote element-by-element powers. X and Y must have the same dimensions unless one is a scalar.

Examples

Let

```
X = [ 1+s 1
      0  1 ];
```

Then

```
X^2
```

returns

```
ans =
      1 + 2s + s^2      2 + s
      0                1
```

On the other hand,

```
X.^[0 1;1 1]
```

returns

Constant polynomial matrix: 2-by-2

`ans =`

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

while

`(s+2).^[1 2 3]`

`ans =`

$$\begin{bmatrix} 2 + s & 4 + 4s + s^2 & 8 + 12s + 6s^2 + s^3 \end{bmatrix}$$

Algorithm

The macros employ basic MATLAB 5 operations.

Diagnostics

The macro `mpower` returns error messages if

- the input matrix is not square
- non-integral or negative powers are encountered

The macro `power` reports an error if

- the matrix dimensions do not agree
- non-integral or negative powers are encountered

See also

`mtimes`

matrix multiplication of polynomial matrices

`times`

array multiplication of polynomial matrices

mtimes (*), times (.*)

Purpose	Matrix multiplication of polynomial matrices
	Element-wise multiplication of polynomial matrices
Syntax	<pre> C = A*B C = mtimes(A,B[,tol]) C = A.*B C = times(A,B[,tol]) </pre>
Description	<p>The commands</p> <pre> C = A*B C = mtimes(A,B) </pre> <p>return the matrix product of the polynomial matrices A and B. Any scalar polynomial may multiply anything. Otherwise, the number of columns of A must equal the number of rows of B. The macro employs zeroing based on the global zeroing tolerance. The command</p> <pre> C = mtimes(A,B,tol) </pre> <p>works with zeroing specified by the relative tolerance <code>tol</code>.</p>

The commands

```
C = A.*B
```

```
C = times(A,B)
```

denote element-by-element multiplication. The polynomial matrices A and B must have the same sizes unless one is a scalar. A scalar can be multiplied into anything. The macro using zeroing based on the global zeroing tolerance. The command

```
C = times(A,B,tol)
```

works with zeroing specified by the relative tolerance `tol`.

Examples

Let

```
X = [ 1+s  1
      0    1 ];
Y = [ 2+s  s^2
      0  3*s^3 ];
```

Then

```
Z1 = X*Y
```

returns

```
z1 =
      2 + 3s + s^2      s^2 + 4s^3
      0                3s^3
```

while

```
z2 = x.*y
```

results in

```
z2 =
      2 + 3s + s^2      s^2
      0                3s^3
```

Note that

```
x*(s+3)^2
```

equals

```
ans =
      9 + 15s + 7s^2 + s^3      9 + 6s + s^2
      0                        9 + 6s + s^2
```

Algorithm

The macros employ basic MATLAB 5 operations.

Diagnostics

The macro `mtimes` returns error messages if

- it does not have enough input arguments
- it has too many output arguments
- the inner matrix dimensions do not agree.

The macro `times` reports an error if the matrix dimensions do not agree.

Both macros issue a warning if the input matrices have inconsistent variables.

See also

`mpower` matrix power of polynomial matrices

`power` array power of polynomial matrices

new2old, old2new

Purpose Convert a polynomial object to and from the format used in versions 1.5 and 1.6 of the Polynomial Toolbox

Syntax

```
Q = new2old(P)

[Q1,Q2,...,Qn] = new2old(P1,P2,...,Pn)

P = old2new(Q)

[P1,P2,...,Pn] = old2new(Q1,Q2,...,Qn)
```

Description In versions 1.5 and 1.6 of the Polynomial Toolbox, which are based on MATLAB version 4, a polynomial matrix

$$P(s) = P_0 + P_1s + \dots + P_d s^d$$

is stored as a 2-dimensional array of the form

$$Q = \left[\begin{array}{cccc|c} P_0 & P_1 & \dots & P_d & d \\ & & & & 0 \\ & & & & \vdots \\ & & & & 0 \\ \hline 0 & 0 & \dots & 0 & \text{NaN} \end{array} \right]$$

The command

```
P = old2new(Q)
```

converts this matrix into a `pol` object – the new format. The command

```
Q = new2old(P)
```

works in the other direction.

Both macros accept several input and output arguments. If the number of output arguments is less than the number of input arguments then the remaining output arguments take over the names of their input counterparts.

Examples

Define in the new format the polynomial

```
P = 1+2*s+3*s^2
```

MATLAB returns

```
P =
```

```
1 + 2s + 3s^2
```

Converting to the old format according to

```
P = new2old(P)
```

we see that

```
P =
```

```
1      2      3      2
```



```
0      0      0      NaN
```

Of course the new format can be re-created

```
P = old2new(P)
```

MATLAB returns

```
P =  
1 + 2s + 3s^2
```

Algorithm

The macros `new2old` and `old2new` use standard MATLAB 5 operations.

Diagnostics

The macros `new2old` and `old2new` return error messages if they do not have enough input arguments.

See also

<code>pol, lop</code>	polynomial matrix constructor
<code>pol2mat, mat2pol</code>	conversion of polynomials and polynomial matrices from Polynomial Toolbox format to MATLAB or Control System Toolbox format and vice-versa
<code>pol2dsp, dsp2pol</code>	conversion of polynomials and polynomial matrices from Polynomial Toolbox format to DSP format and vice-versa

norm

Purpose Norm of a polynomial matrix or a polynomial matrix fraction

Syntax

```
norm(P)
```

```
norm(P,'blk')
```

```
norm(P,'lead')
```

```
norm(P,'max')
```

```
norm(P,type)
```

```
norm(P,'blk',type)
```

```
norm(P,'lead',type)
```

```
norm(P,'max',type)
```

```
norm(N,D)
```

```
norm(N,D,2)
```

```
norm(N,D,'l',2)
```

```
norm(N,D,'r')
```

```
norm(N,D,'r',2)
```

```
norm(N,D,Inf)
```

```
norm(N,D,'l',Inf)
```

```
norm(N,D,'r',Inf)
```

Description

Norms of polynomial matrices

If P is a polynomial matrix with coefficient matrices P_0, P_1, \dots, P_d then

```
norm(P)
```

is the largest singular value of the block coefficient matrix $[P_0 \ P_1 \ \dots \ P_d]$. The number

```
norm(P,'blk')
```

is the same as `norm(P)`. The number

```
norm(P,'lead')
```

is the largest singular value of the leading coefficient matrix, while

```
norm(P,'max')
```

is the greatest of the largest singular values of the coefficient matrices P_0, P_1, \dots, P_d .

The numbers

```
norm(P,TYPE)
```

```
norm(P,'blk',TYPE)
```

equal the norm of type TYPE of the matrix $[p_0 \ p_1 \ \dots \ p_d]$, where TYPE is 1, 2, `Inf`, or `'fro'`. The quantities

```
norm(P,'lead',TYPE)
```

```
norm(P,'max',TYPE)
```

are similarly defined.

Norms for polynomial matrix fractions

Given a stable polynomial matrix D and a polynomial matrix N of compatible dimensions the commands

```
norm(N,D)
```

```
norm(N,D,2)
```

```
norm(N,D,'1',2)
```

compute the H_2 norm of the rational matrix $D^{-1}N$. The commands

```
norm(N,D,'r')
```

```
norm(N,D,'r',2)
```

compute the H_2 norm of the rational matrix ND^{-1} . The commands are synonymous with the command `h2norm`. For more details see the description of `h2norm`.

The calls

```
norm(N,D,Inf)
```

```
norm(N,D,'l',Inf)
```

```
norm(N,D,'r',Inf)
```

return the H_∞ norm of the rational matrix $D^{-1}N$ or ND^{-1} . The command is synonymous to `hinfnorm`. For more details see the description of `hinfnorm`.

Examples

Consider

```
P = [ 1+s  s^2
      3*s  4  ];
```

The block and individual coefficient matrices are

```
P{:}
ans =
    1    0    1    0    0    1
    0    4    3    0    0    0
```

```
P{0}
ans =
    1    0
    0    4
```

```
P{1}
ans =
    1    0
    3    0
```

```
P{2}
ans =
    0    1
    0    0
```

Here are some of the norms:

```
norm(P)
ans =
    5.0400
```

```
norm(P,'max')
```

```
ans =
```

```
4
```

```
norm(P,1)
```

```
ans =
```

```
4
```

For examples of the computation of norms of polynomial matrix fractions see the description of the commands **h2norm** and **hinfnorm**.

Algorithm

The macro uses standard MATLAB and Polynomial Toolbox operations.

Diagnostics

An error message follows if there are too few or too many input arguments, if an unknown command option is encountered, or, in case of the norms for matrix fractions, the matrices have incompatible dimensions or incompatible variables, or the denominator matrix is unstable.

See also

h2norm	computation of the H_2 or H_∞ norm of a stable polynomial
hinfnorm	matrix fraction

null

Purpose Null space of a polynomial matrix

Syntax

```
Z = null(A[,tol])
```

```
Z = null(A,degree[,tol])
```

Description The command

```
Z = null(A)
```

computes a polynomial basis for the right null space of the polynomial matrix A , that is,

$$AZ = 0$$

The basis is minimal in the sense of Forney, that is, Z is column reduced, and has full column rank. Moreover, the columns of Z are arranged according to decreasing degrees. If Y is another polynomial basis then necessarily $\deg(Y, 'col') \geq \deg(Z, 'col')$.

If A has full column rank then Z is an empty polynomial matrix.

The command

```
Z = null(A,degree)
```


seeks a minimal basis assuming that it has degree `degree` or less. If no such basis exists then the empty matrix is returned.

If `degree` is not specified then a minimal polynomial basis is computed by an iterative scheme starting with a basis of degree 0.

If `degree` is negative then the function directly computes a minimal basis without iteration using a computed upper bound for the degree of the basis. This may be more time-consuming than iteration starting with a low degree.

An optional tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Consider

```
A = [ 1+s      s^2      1+s^3      4
      2      3+4*s+s^2      5      6 ];
```

The command

```
z = null(A)
```

yields the null space

```
z =
-1.7 - 5.2s + 2.8s^2 + s^3      -4.6 - 0.14s - 2.1s^2
2.4 - 2s                        -1.7 + 4.2e-016s
```

$$-1.8 - s$$

$$2.1 - 2.1e-016s$$

$$0.88 + 2s$$

$$0.62 + 1.2s + s^2$$

Algorithm

If A is a constant matrix then call the standard MATLAB macro `null` is called. Otherwise first the rank of A is evaluated with the Polynomial Toolbox macro `rank`. Next the null-spaces of Sylvester matrices of A of increasing orders are extracted. The process is terminated when the number of columns of the polynomial null space is equal to the nullity of A that follows from the rank computation.

A description of the algorithm, although specialized to the case of the computation of a coprime right matrix fraction from a left matrix fraction, is provided in Appendix G of Chen (1984).

Diagnostics

The macro issues error messages under the following conditions.

- Invalid `Not-a-Number` or `Inf` entries are encountered in the input matrix
- An invalid argument is found

See also

<code>rank</code>	rank of a polynomial matrix
<code>xab</code>	solution of linear polynomial matrix equations
<code>minbasis</code>	minimal basis

pdg, smith

Purpose Diagonalization of a polynomial matrix

Smith form of a polynomial matrix

Syntax

```
[D,U,V,UI,VI] = pdg(A[,tol])
```

```
[S,U,V,UI,VI] = smith(A[,tol])
```

```
[S,U,V,UI,VI] = smith(A,'elo'[,tol])
```

```
[S,U,V] = smith(A,'moca'[,tol])
```

```
S = smith(A,'zeros'[,tol])
```

Description

The command

```
[D,U,V,UI,VI] = pdg(A,tol)
```

converts the polynomial matrix A to a diagonal matrix D by elementary operations so that $UAV = D$. U and V are unimodular transformation matrices. The inverse matrices $UI = U^{-1}$ and $VI = V^{-1}$ are also computed. A tolerance `tol` may be specified as an additional input argument.

The command

```
[S,U,V,UI,VI] = smith(A)
```

computes the Smith form S of the polynomial matrix A . U and V are unimodular matrices such that $UAV = S$. A polynomial matrix S is in Smith form if it is given by

$$S = \begin{bmatrix} s_1 & 0 & \cdots & \cdots & 0 \\ 0 & s_2 & 0 & & 0 \\ \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & \cdots & 0 & s_k & 0 \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix}$$

The integer k is the rank of S and the polynomial diagonal entries satisfy the condition that s_j divides s_{j+1} for $j = 1, 2, \dots, k-1$.

The command

```
[S,U,V,UI,VI] = smith(A,'elo')
```

computes the Smith form by elementary operations. This is the default method. The command

```
[S,U,V] = smith(A,'moca')
```

computes the Smith form of A by a “Monte Carlo” method. The call

```
S = smith(A,'zeros')
```

finally, returns the Smith diagonal form of a nonsingular polynomial matrix A based on the computation of the zeros of A and their algebraic and geometric multiplicities. The option '**zeros**' can only be used for square matrices.

U and V are unimodular transformation matrices and their inverse matrices are $UI = U^{-1}$ and $VI = V^{-1}$. The “Monte Carlo” algorithm (the option '`moca`') does not return UI and VI . The method based on zeros (the option '`zeros`') does not return any of the unimodular matrices U , V , UI and VI .

A tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Consider

$$A = \begin{bmatrix} 1+s & 0 & 0 \\ 0 & 2+3s+s^2 & 0 \\ 0 & 2+s & 4+4s+s^2 \end{bmatrix};$$

Then the command

$$D = \text{pdg}(A)$$

yields the diagonal form

$$D = \begin{bmatrix} 1 + s & 0 & 0 \\ 0 & -2 - s & 0 \\ 0 & 0 & -4 - 8s - 5s^2 - s^3 \end{bmatrix}$$

Inspection shows that D is not in Smith form. The Smith form of A follows as

```
S = smith(A)
```

```
S =
```

```

-1.2      0      0
      0    -1.7 - 2.5s - 0.85s^2    0
      0      0      -4 - 8s - 5s^2 - s^3

```

We check that the (3,3) entry divides the (2,2) entry by typing

```
S(3,3)/S(2,2)
```

```
ans =
```

```
2.4 + 1.2s
```

Algorithm

Pdg

The matrix is reduced both row-wise and column-wise by a series of elementary column and row operations (Kucera, 1979). At each step, the least degree nonzero polynomial is found in the input matrix and then brought to the upper left-hand corner by suitable row and column interchanges. Next the entry is used to reduce all the elements in the first row and column. This process is repeated until all the polynomials in the first row and column, other than the upper left-hand corner element, are reduced to zero. Repetition of the above steps on the remaining rows and columns produces a diagonal matrix.

Smith

Option '**e1o**' (default)

The matrix is reduced both row-wise and column-wise by a series of elementary column and row operations (Kucera, 1979). At each step the least degree nonzero polynomial is found in the input matrix and brought to the upper left-hand corner by suitable row and column interchanges. After this the entry is used to reduce all the elements in the first row and column. This process is repeated until all the polynomials in the first row and column, other than the upper left-hand corner element, are reduced to zero. Repetition of the above steps for the remaining rows and columns produces a diagonal matrix. For every new k th diagonal element it is checked whether $a_{k-1,k-1}$ divides a_{kk} . If this the case then the algorithm can proceed. If not then a greatest common divisor g of the two polynomials is calculated with the help of **gcd** and expressed in the form

$$a_{k-1,k-1}p + a_{kk}q = g$$

The $(k-1)$ st and k th rows and columns are transformed by elementary row and column operations to bring the matrix into the form

$$\begin{bmatrix} a_{11} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & 0 & \cdots & \cdots & \cdots & 0 \\ \cdots & 0 & g & a_{k-1,k-1} & 0 & \cdots & 0 \\ 0 & \cdots & a_{kk} & 0 & 0 & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & \times & \cdots & \times \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & \times & \cdots & \times \end{bmatrix}$$

The major steps of the algorithm then are recommenced on the $(k-1)$ st row and column.

Option 'moca'

The option '**moca**' uses a “Monte Carlo” method. These are the steps of the algorithm:

1. Compute the Hermite form of the input matrix
2. Compute the Hermite form of the transpose of the output of the previous step
3. If the output of step 2 is not diagonal then repeat step 2
4. If the output is diagonal check if $a(k-1, k-1)$ divides $a(k, k)$
5. If the output is not diagonal, or if $a(k-1, k-1)$ does not divide $a(k, k)$ then repeat the steps 1 to 4 once more after pre- and postmultiplying the input matrix by two random unimodular matrices

6. If also the second complete run produces no solution then the computation is cancelled and an error message is returned

Using this method we obtain only transformation matrices U and V (but not their inverses because the macro **hermite** does not return these inverses).

Option 'zeros'

If the '**zeros**' option is activated then the macro first computes the zeros of A with the macro **roots** and then evaluates their algebraic and geometric multiplicities with the macro **character**. The transformation matrices are not found this way and the method applies to square matrices only.

Diagnostics

The macros issue error messages if

- an invalid option is encountered (**smith** only)
- an invalid tolerance is specified
- the input matrix is not square ('**zeros**' option of **smith** only)
- the reduction fails in any of several ways

See also

hermite Hermite form of a polynomial matrix

pencan

Purpose Transformation of a nonsingular matrix pencil to Kronecker canonical form

Syntax

```
[C,Q,Z,dims] = pencan(P)
[C,Q,Z,dims] = pencan(P,tol)
[C,Q,Z,dims] = pencan(P,'ord')
[C,Q,Z,dims] = pencan(P,'ord',tol)
```

Description The command

```
[C,Q,Z,dims] = pencan(P)
```

transforms the square nonsingular real pencil

$$P(s) = P_0 + P_1 s$$

to the real Kronecker canonical form

$$Q(s) = QP(s)Z = \begin{bmatrix} a + sI & 0 \\ 0 & I + se \end{bmatrix}$$

The matrix a is in real Schur form, that is, it is upper block triangular with the real eigenvalues on the diagonal and the complex eigenvalues in 2x2 blocks on the

diagonal. The real matrix e is upper triangular with zeros on the diagonal and, hence, is nilpotent. Q and Z are real nonsingular.

If the option `'ord'` is included then a is in block diagonal form $a = \text{diag}(a1, a2)$, with $a1$ and $a2$ both in real Schur form. All eigenvalues of $a1$ have strictly positive real parts and those of $a2$ have non-positive real parts.

The output argument `dims` contains the sizes `[na ne]` of a and e . If the option `'ord'` is included then `dims` contains the sizes `[na1 na2 ne]` of $a1$, $a2$ and e .

The number `tol` is an optional tolerance that is used to decide which eigenvalues of the pencil are infinite. Its default value is `eps`.

If the verbose property is enabled then the macro reports the relative error.

Examples

Consider the pencil

$$P = \begin{bmatrix} s & -1 & 0 & 0 & 0 & 0 \\ 0 & s & 0 & 0 & 0 & 0 \\ 0 & 0 & 2+s & -1 & 0 & 0 \\ 0 & 0 & 0 & s & 0 & 0 \\ 0 & 0 & 0 & 0 & s & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix};$$

We compute its Kronecker canonical form:

```
[C,Q,Z,dims] = pncan(P); C, dims
```

```
C =
```

```

      s      -1      0      0      0      0
      0      s      0      0      0      0
      0      0      2 + s    -1      0      0
      0      0      0      s      0      0
      0      0      0      0      1      s
      0      0      0      0      0      1

```

```
dims =
```

```

      4      2

```

Inspection shows that we have

$$a = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad e = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

“Ordered” transformation yields

```
[C,Q,Z,dims] = pncan(P,'ord'); C, dims
```

```

C =
      2 + s      0      0      0      0      0
      0          s      0      0      0      0
      0          0      s     -1      0      0
      0          0      0      s      0      0
      0          0      0      0      1      s
      0          0      0      0      0      1

dims =
      1      3      2

```

Hence for the ordered form we have

$$a = \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \end{bmatrix}, \quad a_1 = 2, \quad a_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}, \quad e = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Algorithm

Using ordered QZ transformation (see `qzord`) the two coefficient matrices `P0` and `P1` of P are simultaneously transformed to upper triangular form. The ratios of the diagonal entries of the transformed `P0` and `P1` that are infinite come last. If the option `ord` is used in `qzord` then the finite ratios are ordered according to decreasing real parts. After this the transformed pencil is block diagonalized by

“nulling” the off-diagonal blocks with the help of `plyap`. The transformation is completed by first making the result real and then transforming the various diagonal blocks with the assistance of `schur` so that they assume the desired forms.

Diagnostics

The macro displays error messages in the following situations:

- the first input argument is not a square pencil
- the pencil is not real
- there are too many input arguments
- an unknown option is encountered

A warning message is issued if the relative residue is greater than $1e-6$. The relative residue is defined as the norm of $C-QPZ$ divided by the norm of P . For the purpose of this test the norm of a pencil is defined as the largest of the norms of its coefficient matrices.

See also

<code>plyap</code>	solution of a two-sided linear pencil equation
<code>qzord</code>	ordered QZ transformation

pinit, gprop, pformat, gensym, tolerance, verbose

Purpose Initialize the Polynomial Toolbox

Set global properties

Syntax

`pinit`

`pinit GlobalPropertyValue`

`pinit GlobalPropertyValue1 GlobalPropertyValue2 ...`

`gprop`

`gprop GlobalPropertyValue`

`gprop GlobalPropertyValue1 GlobalPropertyValue2 ...`

`pformat`

`pformat OutputFormat`

`gensym`

`gensym DefaultSymbol`

`tolerance`

`tolerance ToleranceValue`

verbose

verbose no

verbose yes

Description

The macro **pinit** must be called at the beginning of every Polynomial Toolbox session. Typing

pinit

without input arguments sets the global properties equal to their default values. If the command is followed by a blank space separated list of admissible values of global properties then the corresponding global properties are set equal to the prescribed values while the other properties remain at their defaults.

If used without input arguments the command

gprop

returns the list of current values of the global polynomial properties and the admissible alternatives. The macro may also be used to change one or more global polynomial properties to new admissible values that are specified in the blank space separated list following the command. The new values then are displayed.

The **pformat** command switches between different display formats for polynomial matrices.

<code>pformat</code>	same as <code>pformat coef</code>
<code>pformat coef</code>	matrix coefficients in order of ascending power
<code>pformat rcoef</code>	matrix coefficients in order of descending powers
<code>pformat block</code>	block row composite coefficient matrix
<code>pformat symb</code>	symbolic format.
<code>pformat syms</code>	symbolic format with short coefficients.
<code>pformat symbr</code>	symbolic format with rational coefficients (expressed in the rational format of MATLAB).

In all “non-symbolic” polynomial formats the format of the scalar coefficients is governed by the standard MATLAB command `format`.

The command `gensym` is used to set the default variable symbol.

<code>gensym</code>	default; same as <code>gensym s</code>
<code>gensym s</code>	variable for continuous-time operator
<code>gensym p</code>	variable for continuous-time operator
<code>gensym z</code>	variable for discrete-time operator
<code>gensym q</code>	variable for discrete-time operator
<code>gensym d</code>	variable for discrete-time backward shift(delay) operator

gensym z^{-1} variable for discrete-time backward shift(delay) operator

Similarly, the command **tolerance** sets the global tolerance as follows:

tolerance default; same as tolerance **1e-8**

tolerance 1e-8 set the global tolerance equal to **1e-8**

tolerance (1e-8) set the global tolerance equal to **1e-8**

tolerance tol set the global tolerance equal to **tol**

tolerance (tol) set the global tolerance equal to **tol**

Here **tol** is any real number.

The **verbose** command is used to switch between different global verbose level as follows:

verbose default; same as **verbose no**

verbose no no comments during execution

verbose yes provide comments during execution

Examples

After starting up MATLAB, typing

pinit

provokes the response

Polynomial Toolbox initialized. To get started, type one of these: `helpwin` or `poldesk`. For product information, visit www.polyx.com or www.polyx.cz.

The command only works, however, if the Polynomial Toolbox directory is in the MATLAB search path. The default global properties may be viewed after typing

`gprop`

Global polynomial properties:

PROPERTY NAME:	CURRENT VALUE:	AVAILABLE VALUES:
variable symbol	s	's','p','z'^-
1','d','z','q'		
zeroing tolerance	1e-008	any real number
verbose level	no	'no', 'yes'
display format	syms	'symb', 'syms', 'symbr'
		'coef', 'rcoef', 'block'

The default variable is changed to z by the command

`gprop z`

We may also change the display format, default symbol (once again), and the tolerance, for instance by

```
pformat coef, gensym d, tolerance eps
```

To inspect the changes one may type once again

```
gprop
```

```
Global polynomial properties:
```

PROPERTY NAME:	CURRENT VALUE:	AVAILABLE VALUES:
variable symbol 1','d','z','q'	d	's','p','z^-'
zeroing tolerance	2.2204e-016	any real number
verbose level	no	'no', 'yes'
display format	coef	'symb', 'syms', 'symbr' 'coef', 'rcoef', 'block'

Finally, the command

```
pinit 1e-6 z^-1
```

is a shortcut for

pinit, gprop, pformat, gensym, tolerance, verbose

```
pinit
```

```
gprop 1e-6 z^-1
```

Algorithm

The macros use standard MATLAB 5 commands.

Diagnostics

The macros return error messages if the input is inappropriate.

See also

<code>pprop</code>	set or modify the properties of a polynomial matrix
<code>symbol</code>	return or modify the variable of a polynomial matrix
<code>pol</code>	create a polynomial matrix object

pinv

Purpose Pseudo-inverse of a full-rank polynomial matrix

Syntax

```
[Q,d] = pinv(P[,tol])
[Q,d] = pinv(P,'int'[,tol])
[Q,d] = pinv(P,'def'[,tol])
```

Description Given an $n \times m$ full rank polynomial matrix P the command

```
[Q,d] = pinv(P)
```

produces an $m \times n$ polynomial matrix Q and a scalar monic polynomial d such that the rational matrix Q/d is the Moore-Penrose pseudo-inverse of the matrix P , that is,

$$PQP = Pd$$

$$QPQ = Qd$$

Moreover, PQ and QP are symmetric, that is, $(PQ)^T = PQ$, $(QP)^T = QP$.

The default method is `'int'`, and relies on the `inv` routine with the interpolation option. The option `'def'` calls the corresponding “definition” option of `inv`.

The optional input parameter `tol` is a relative tolerance for zeroing and rank testing. Its default value is the global zeroing tolerance, except in rank testing, where the default value is the MATLAB default.

Examples

Consider the polynomial matrix

```
P = [1+s s^2 3];
```

The command

```
[Q,d] = pinv(P)
```

results in

```
Q =
```

```
1 + s
```

```
s^2
```

```
3
```

```
d =
```

```
10 + 2s + s^2 + s^4
```

It is easy to check that the various properties hold.

Given the polynomial equation $Px = a$, with P wide full rank and a a given polynomial vector, the pseudo inverse may be used to determine a rational solution x . For the given matrix P , for instance, taking $a = 1$ results in the rational solution

$$x = Qa/d$$

where

```
a = 1; Q*a
```

```
ans =
```

```
1 + s
```

```
s^2
```

```
3
```

Algorithm

The pseudo-inverse Q is defined as follows:

- If P is wide then $Q/d = P^T(PP^T)^{-1}$
- If P is tall then $Q/d = (P^TP)^{-1}P^T$
- If P is square then $Q/d = P^{-1}Q = P$

The inverses are computed using `inv`.

Diagnostics

The macro returns an error message if the input matrix does not have full rank to working precision.

See also

`inv` inverse of a polynomial matrix

plqg

Purpose Polynomial solution of LQG problems

Syntax

```
[Nc,Dc] = plqg(N,D,Q1,R1,Q2,R2)
[Nc,Dc] = plqg(N,D,Q1,R1,Q2,R2,'l')
[Nc,Dc] = plqg(N,D,Q1,R1, Q2,R2,'r')
```

Description Consider a linear time-invariant plant with transfer matrix

$$P(v) = D^{-1}(v)N(v)$$

where v can be any of the variables s, p, z, q, z^{-1} or d . The commands

```
[Nc,Dc] = plqg(N,D,Q1,R1,Q2,R2)
[Nc,Dc] = plqg(N,D,Q1,R1,Q2,R2,'l')
```

compute an LQG optimal regulator as in Fig. 6 with transfer matrix

$$Q(v) = N_C(v)D_C^{-1}(v)$$

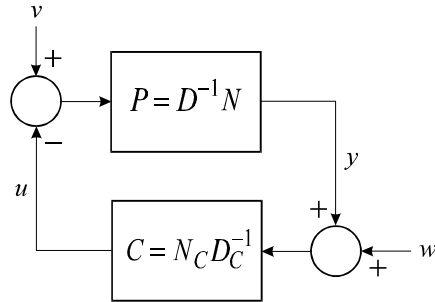


Fig. 6. LQG feedback structure

The controller minimizes the steady-state value of

$$E(y^T(t)Q_2y(t) + u^T(t)R_2u(t))$$

Q_2 and R_2 are symmetric nonnegative-definite weighting matrices. The symmetric nonnegative-definite matrices Q_1 and R_1 represent the intensities (covariance matrices) of the input and measurement white noises, respectively, and need to be nonsingular.

Examples

Example 1 (Kucera, 1991, pp. 298–303). Consider the discrete-time plant described by the left matrix fraction $D^{-1}(z)N(z)$, where

$$N = [1; 1]$$

$$D = [z^{-1/2} \ 0; 0 \ z^2]$$

Define the nonnegative-definite matrices

```
Q1 = 1;
Q2 = [7 0; 0 7];
R1 = [0 0; 0 1];
R2 = 1;
```

The optimal LQG controller is obtained by typing

```
[Nc,Dc] = plqq(N,D,Q1,R1,Q2,R2)
```

MATLAB returns

```
Nc =
    -0.10633z^2      0

Dc =
    -0.21266z^2 - 0.85065z^3      0
    0.10633 + 0.34409z^2 - 1.3764z^3    1.1756
```

If the same plant is described by a right matrix fraction description $N(z)D^{-1}(z)$, with

```
N = [z^2; z-1/2]
D = z^2*(z-1/2)
```

then the controller results by typing

```
[Nc,Dc] = plqg(N,D,Q1,R1,Q2,R2, 'r')
```

Constant polynomial matrix: 1-by-2

Nc =

0.5 0

Dc =

1 + 4z

Example 2. Consider a continuous-time problem described by

```
N = [1; 1]
```

```
D = [s-2 0; 0 s^2+1]
```

and the weighting matrices

```
Q1 = 1;
```

```
R1 = eye(2);
```

```
Q2 = [10 0; 0 2];
```

```
R2 = 1;
```

The call

```
[Nc,Dc] = plqg(N,D,Q1,R1,Q2,R2)
```

returns the optimal LQG feedback controller

```
Nc =
```

```
-1.3436 + 0.94635s      17.0728
```

```
Dc =
```

```
-0.61922 + 0.12144s      5.0345 + s
```

```
3.2092 + 2.5541s + s^2      -7.5191 - 16.4344s
```

The closed-loop poles of the optimal feedback system follow as

```
roots(N*Nc+D*Dc)
```

```
ans =
```

```
-3.72972478386939
```

```
-2.22996241633438
```

```
-0.42364369708415 + 1.07974078889846i
```

```
-0.42364369708415 - 1.07974078889846i
```

```
-0.38868587951944 + 1.05190312987575i
```

```
-0.38868587951944 - 1.05190312987575i
```

Algorithm	The computation involves two spectral factorizations and the solution of a Diophantine equation solution. The algorithm is derived in Kucera (1991, pp. 306–310).
Diagnostics	<p>The macro displays error messages in the following situations:</p> <ul style="list-style-type: none"> • There are not enough input arguments • The denominator matrix is not square • The input arguments have inconsistent sizes • The plant is not proper • The covariance matrices are not square symmetric • The weighting matrices are not square symmetric • R_1 or R_2 is not positive-definite
See also	<code>sp1qg</code> scalar LQG design

plyap

Purpose Solution of a two-sided equation with matrix pencil coefficients

Syntax `[X,Y] = plyap(A,B,C)`
`[X,Y] = plyap(A,B,C,tol)`

Description Given square matrix pencils $A(s) = A_0 + A_1s$ and $B(s) = B_0 + B_1s$ (that is, polynomial matrices of degree 1) and a pencil of consistent dimensions $C(s) = C_0 + C_1s$ the macro returns constant matrices X and Y that are the solution of

$$A(s)X + YB(s) = C(s)$$

The equation has a unique solution if and only if the pencils A and B have no common zeros (including zeros at infinity). If this condition is not satisfied then the solution may be inaccurate.

The optional parameter `tol` is a tolerance that is used in determining whether A or B is upper or lower triangular. Its default value is `sqrt(eps)`.

If the verbose property is enabled then the macro reports the relative residue, which is defined as the ratio of the norm of $A(s)X + YB(s) - C(s)$ to the largest of the norms of A , B and C . For the purpose of this test the norm of a polynomial matrix is defined as the norm of its composite coefficient matrix.

Examples

Consider the pencil Lyapunov equation with coefficient matrices

```
A = [ 3    7    9
      11   -1   -19
           1   20    0];

B = [-3+3*s    -10+7*s
      4+15*s    -11-6*s];

C = [-19+10*s    3
      6+8*s      17
      -4         -2+6*s];
```

The command

```
[X,Y] = plyap(A,B,C)
```

results in the output

```
X =
    -2.1411    3.1965
     0.0461    0.0255
    -1.5236    0.4855
```



```
Y =
    0.4878    0.5691
    0.3902    0.4553
    0.7317   -0.1463
```

If the matrix B is modified to

```
B = [-3+3*s    -10
      4+15*s   -11];
```

then A has two zeros at infinity and B one zero at infinity, and the computation fails:

```
[X,Y] = plyap(A,B,C)
```

```
Warning: Matrix is singular to working precision.
```

```
plyap warning: Solution has Not-a-Number or Infinite
entries
```

```
X =
    Inf    Inf
    Inf    Inf
    Inf    Inf
```

```

Y =

    Inf    -Inf
    NaN     NaN
    NaN     NaN

```

Algorithm

With the help of the QZ algorithm (see `qz`) the pencil B is transformed to upper triangular form. When this has been done the resulting equations may be solved for X and Y column by column from the left to the right.

Before transforming B it is checked whether A or B is upper or lower triangular. If this is the case then the equation may be rearranged so that B is upper triangular without the QZ transformation. For the triangularity test the tolerance `tol` is used.

Diagnostics

The macro displays error messages in the following situations :

- The pencil A is not square
- The pencil B is not square
- C has inconsistent dimensions

A warning message is issued if

- The solution has `Not-a-Number` or `Infinite` entries
- The relative residue is greater than `1e-6`

See also

`pencan`

transformation of a matrix pencil to Kronecker canonical form

pme

Purpose Polynomial Matrix Editor

Syntax `pme`

Description The Polynomial Matrix Editor (PME) is recommended for creating and editing polynomial and standard MATLAB matrices of medium to large size, say from about 4×4 to 30×35 . Matrices of smaller size can easily be handled in the MATLAB command window with the help of monomial functions, overloaded concatenation, and various applications of subscripting and subassigning. On the other hand, opening a matrix larger than 30×35 in the PME results in a window that is difficult to read.

Type `pme` to open the main window called Polynomial Matrix Editor. This window displays all polynomial matrices (POL objects) and all standard MATLAB that exist in the main MATLAB workspace. It also allows you to create a new polynomial or standard MATLAB matrix. In the Polynomial Matrix Editor window you can

- create a new polynomial matrix, by typing its name and size in the first (editable) line and then clicking the `Open` button
- modify an existing polynomial matrix while retaining its size and other properties. To do this just find the matrix name in the list and then double click the particular row

Commands — Online reference

- modify an existing polynomial matrix to a large extent (for instance by changing also its name, size, variable symbol, etc.): To do this, first find the matrix name in the list and then click on the corresponding row to move it up to the editable row. Next type in the new required properties and finally click **Open**

Each of these actions opens another window called Matrix Pad that serves for editing the matrix.

In the Matrix Pad window the matrix entries are displayed as boxes. If an entry is too long so that it cannot be completely displayed then the corresponding box takes a slightly different color (usually more pinkish.)

To edit an entry just click on its box. The box becomes editable and large enough to display its entire content. You can type into the box anything that complies with the MATLAB syntax and that results in a scalar polynomial or constant. Of course you can use existing MATLAB variables, functions, etc. The program is even more intelligent and handles some notation going beyond the MATLAB syntax. For example, you can drop the * (times) operator between a coefficient and the related polynomial symbol (provided that the coefficient comes first). Thus, you can type **2s** as well as **2*s**.

To complete editing the entry push the **Enter** key or close the box by using the mouse. If you have entered an expression that cannot be processed then an error message is reported and the original box content is recovered.

To bring the newly created or modified matrix into the MATLAB workspace finally click **Save** or **Save As**.

Algorithm

The macro uses standard MATLAB 5 commands.

See also

`pol` create a polynomial matrix object

pol, lop**Purpose**

Create a polynomial matrix

Convert a constant matrix or a matrix in symbolic toolbox format to a polynomial matrix

Syntax

`P = pol(A,d)`

`P = pol(A,d,PropertyValue1,...)`

`P = pol(Q)`

`P = lop(A,d)`

`P = lop(A,d,PropertyValue1,...)`

`P = lop(Q)`

Description

Given a constant matrix $A = [P_0 \ P_1 \ P_2 \ \dots \ P_d]$ the command

`P = pol(A,d)`

creates a polynomial matrix object P that defines the polynomial matrix

$$P(s) = P_0 + P_1 s + P_2 s^2 + \dots + P_d s^d$$

As further arguments assignable Property/Value arguments may be listed, in particular the variable of the polynomial matrix. If none are listed then the default values are chosen. Type `help pprop` for details on assignable properties.

If not specified by a Property/Value argument then the variable string v copies the current value of the global variable `symbol`. The polynomial matrix variable string may be

- the continuous-time operator s (default) or p
- the discrete-time forward shift operator z or q
- the discrete-time backward shift (delay) operator d or z^{-1}

If A is a zero matrix and/or $d = -\text{Inf}$ then P is a zero polynomial matrix. If A and/or d is empty then an empty polynomial matrix object is created.

The command

```
P = pol(Q)
```

returns $P = Q$ if Q is already a polynomial matrix object. If Q is a standard MATLAB constant matrix, then the command returns the corresponding zero degree polynomial matrix object.

If Q is a symbolic matrix of the Symbolic Toolbox, then

```
P = pol(Q)
```


converts Q to a polynomial matrix P . The variable string copies the current value of the global variable symbol.

The command

$$P = \text{lop}(A, d)$$

expects the compound coefficient matrix A to be arranged in the reversely ordered form

$$A = [Pd \dots P2 \ P1 \ P0]$$

Examples

We create the 2×2 polynomial matrix

$$P(s) = \begin{bmatrix} 1+s+2s^2 & -s \\ -3+4s & 5s^2 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ -3 & 0 \end{bmatrix}}_{P0} + \underbrace{\begin{bmatrix} 1 & -1 \\ 4 & 0 \end{bmatrix}}_{P1}s + \underbrace{\begin{bmatrix} 2 & 0 \\ 0 & 5 \end{bmatrix}}_{P2}s^2$$

of degree 2 in the variable s by the commands

$$P0 = [1 \ 0; \ -3 \ 0];$$

$$P1 = [1 \ -1; \ 4 \ 0];$$

$$P2 = [2 \ 0; \ 0 \ 5];$$

$$P = \text{pol}([P0 \ P1 \ P2], 2)$$

This results in

```
P =
      1 + s + 2s^2      -s
      -3 + 4s          5s^2
```

We redefine P as a polynomial matrix in the variable z by the command

```
P = pol([P0 P1 P2],2,'z')
```

so that

```
P =
      1 + z + 2z^2      -z
      -3 + 4z          5z^2
```

Note how setting the variable property to `z` causes the result to be displayed as a polynomial matrix in the variable z .

Finally, typing

```
P = pol(zeros(1,3)), Q = pol(zeros(0,3))
```

results in

```
Zero polynomial matrix: 1-by-3, degree: -Inf
P =
```

0 0 0

$Q =$

Empty polynomial matrix: 0-by-3

Algorithm

The macros use standard MATLAB 5 commands.

Diagnostics

The macro returns error messages if the input is incorrect or inconsistent.

See also

`pprop` set or modify the properties of a polynomial matrix

pol2dsp

Purpose Conversion of a polynomial or polynomial matrix to DSP format

Syntax `M = pol2dsp(P)`

Description The command

`M = pol2dsp(P)`

converts the polynomial or polynomial matrix P in Polynomial Toolbox format to DSP format.

In DSP format a polynomial is represented as a row vector whose entries are the coefficients of the polynomial according to ascending powers. A polynomial matrix is a cell array of the same dimensions as the polynomial matrix, with each cell a row vector representing the corresponding entry of the polynomial matrix in DSP format. The DSP format is typically used for discrete-time systems.

Examples *Conversion of a (scalar) polynomial:*

```
P = 1+2*q+3*q^2;
```

```
M = pol2dsp(P)
```

```
M =
```

```
      1      2      3
```

Conversion of a polynomial matrix:

```
P = [ 1+2*q 3+4*q^2 ];
M = pol2dsp(P)

M =

      [1x3 double]      [1x3 double]
```

View the entries:

```
M{1,1}, M{1,2}

ans =

      1      2      0

ans =

      3      0      4
```

Algorithm

The macro `pol2dsp` uses standard MATLAB 5 operations.

Diagnostics

The macro `pol2dsp` displays no error messages.

See also

<code>dsp2pol</code>	conversion from DSP format to Polynomial Toolbox format
<code>pol2mat</code>	conversion from Polynomial Toolbox format to MATLAB or Control System Toolbox format

mat2pol conversion from MATLAB or Control System Toolbox format to Polynomial Toolbox format

pol2mat

Purpose	Conversion of a polynomial or polynomial matrix to MATLAB or Control System Toolbox format
Syntax	$\mathbf{M} = \text{pol2mat}(\mathbf{P})$ $\mathbf{M} = \text{pol2mat}(\mathbf{P}, 'leg')$
Description	<p>The command</p> $\mathbf{M} = \text{pol2mat}(\mathbf{P})$ <p>converts the polynomial or polynomial matrix \mathbf{P} in Polynomial Toolbox format to MATLAB or Control System Toolbox format.</p> <p>In MATLAB format a polynomial is represented as a row vector whose entries are the coefficients of the polynomial according to descending powers. In Control System Toolbox format a polynomial matrix is a cell array of the same dimensions as the polynomial matrix, with each cell a row vector representing the corresponding entry of the polynomial matrix in MATLAB format. If the polynomial matrix consists of a single element then in Control System Toolbox format it is represented as a row vector with entries in MATLAB format.</p> <p>To comply with an older standard in the Control System Toolbox, polynomial matrices consisting of a single column may be represented as a matrix whose rows</p>

are the polynomial entries in MATLAB format. The macro `pol2mat` converts to this older format if the legacy option `'leg'` is used.

Examples

Conversion of a (scalar) polynomial:

```
P = 1+2*s+3*s^2;
```

```
M = pol2mat(P)
```

```
M =
```

```
      3      2      1
```

Conversion of a polynomial matrix:

```
P = [ 1+2*s 3+4*s^2 ];
```

```
M = pol2mat(P)
```

```
M =
```

```
      [1x3 double]      [1x3 double]
```

View the entries:

```
M{1,1}, M{1,2}
```

```
ans =
```

```
      0      2      1
```



```
ans =  
      4      0      3
```

However, using the legacy option,

```
Q = [ 1+2*s  
      3+4*s^2 ];
```

is converted to

```
pol2mat(Q,'leg')  
  
ans =  
      0      2      1  
      4      0      3
```

Algorithm

The macro `pol2mat` uses standard MATLAB 5 operations.

Diagnostics

The macro `pol2mat` displays no error messages.

See also

<code>mat2pol</code>	conversion from MATLAB or Control System Toolbox format to Polynomial Toolbox format
<code>pol2dsp</code>	conversion from Polynomial Toolbox format to DSP format
<code>dsp2pol</code>	conversion from DSP format to Polynomial Toolbox format

pol2root, root2pol

Purpose Extract zeros and gains from a polynomial matrix

Construct a polynomial matrix from its zeros and gains

Syntax `[Z,K] = pol2root(P)`

`P = root2pol(Z,K[,var])`

`P = root2pol(Z[,var])`

Description The command

`[Z,K] = pol2roots(P)`

returns a cell array Z that contains the roots and a two-dimensional array K that contains the gains of the entries of the polynomial matrix P .

Conversely, given a cell array Z that contains the roots and a two-dimensional array K that contains the gains of the entries of a polynomial matrix the command

`P = root2pol(Z,K)`

returns the polynomial matrix P . If K is missing then it is supposed to be a ones-array.

The commands

```
P = root2pol(Z,K,VAR)
```

```
P = root2pol(Z,VAR)
```

allow the user to specify the variable of P .

Examples

Consider the polynomial matrix

```
P = [ 1+2*s   s^2/3   1+s+s^2
      0       1+s^3    2       ];
```

The roots and gains follow as

```
[Z,K] = pol2root(P)
```

MATLAB returns

```
Z =
    [-0.5000]    [2x1 double]    [2x1 double]
           []    [3x1 double]           []

K =
    2.0000    0.3333    1.0000
         0    1.0000    2.0000
```

Inspection of the (1,2)th entry of Z by typing

```
z{1,2}
```

yields the expected result

```
ans =
```

```
0
```

```
0
```

We recreate the polynomial matrix, but now in the variable p , by the command

```
Q = root2pol(Z,K,'p')
```

```
Q =
```

```

      1 + 2p      0.33p^2      1 +
p + p^2
      0      1 + 3.3e-016p - 5.6e-016p^2 + p^3      2
```

Algorithm

The macros `pol2root` and `root2pol` use standard MATLAB 5 and Polynomial Toolbox operations.

Diagnostics

The macro `root2pol` returns error messages under the following circumstances

- the roots do not form a cell array
- the roots and gains do not have matching dimensions
- the zeros are not represented as a vector

See also

<code>pol</code>	polynomial matrix constructor
<code>roots</code>	roots of a polynomial matrix
<code>zpk2lmf</code> , <code>zpk2rmf</code> <code>lmf2zpk</code> , <code>rmf2zpk</code>	conversion from Control System Toolbox <code>zpk</code> format to a left or right matrix fraction, and vice-versa

polblock

Purpose Create a Simulink block for a polynomial matrix fraction

Syntax `polblock`

Description Type

`simulink`

to run SIMULINK. When the Blocksets & Toolboxes icon is double-clicked then the Polynomial Toolbox icon automatically appears as in Fig. 7, provided that the Polynomial Toolbox directory is included in the MATLAB search path.

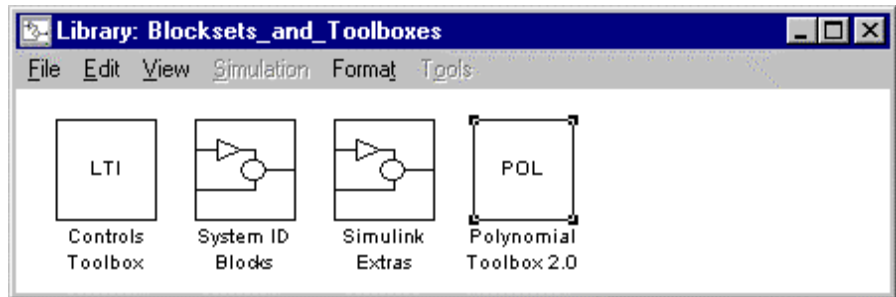


Fig. 7. Blocksets and Toolboxes Library window

By double-clicking the Polynomial Toolbox icon the Polynomial Library window is opened. Alternatively, typing

`polblock`

opens the Polynomial Library window (Fig. 8) directly.

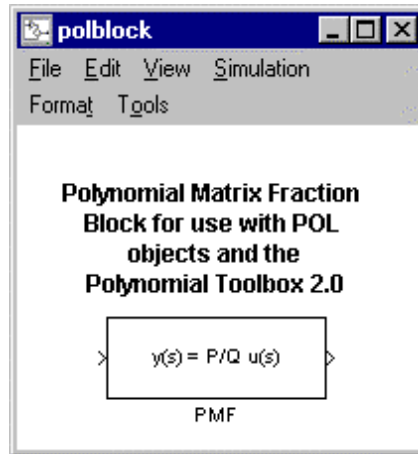
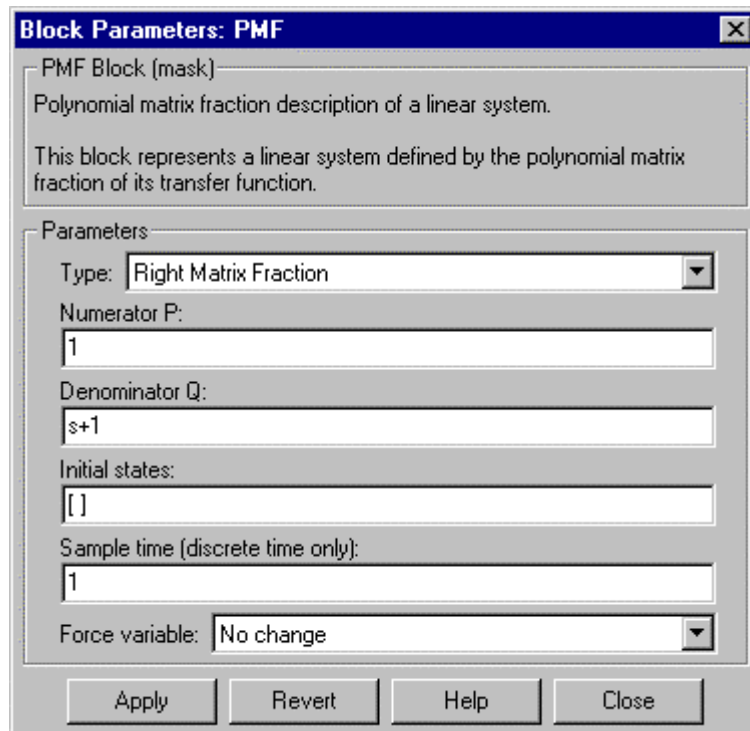


Fig. 8. Polynomial Matrix Fraction Library window

The Polynomial Fraction Block can be handled as any other SIMULINK built-in blocks to form desired models. By double-clicking the icon the window of Fig. 9 or a window like it appears.



Block Parameters: PMF

PMF Block (mask)

Polynomial matrix fraction description of a linear system.

This block represents a linear system defined by the polynomial matrix fraction of its transfer function.

Parameters

Type:

Numerator P:

Denominator Q:

Initial states:

Sample time (discrete time only):

Force variable:

Apply Revert Help Close

Fig. 9. PMF parameters window

The window is ready for directly typing in the matrices. Alternatively, given two polynomial matrices P and Q in the MATLAB workspace, you may just type P and Q into the dialogue prompts of this window.

The variable of the polynomial matrices determines whether the system is continuous-time (s and p) or discrete-time (z , q , d or z^{-1}). The variable may be overwritten by setting it in the “Force variable” window.

Example

The block parameters window of Fig. 9 defines the numerator and denominator polynomials

$$P(s) = 1, \quad Q(s) = 1 + s$$

and, hence, describes the SISO system with transfer function

$$H(s) = \frac{1}{1 + s}$$

After closing the block parameters input window the Matrix Fraction icon may be dragged to the system model window. Adding “Step” and “Scope” blocks we easily construct the simple block diagram of Fig. 10.

Running the simulation yields the expected output (Fig. 11).

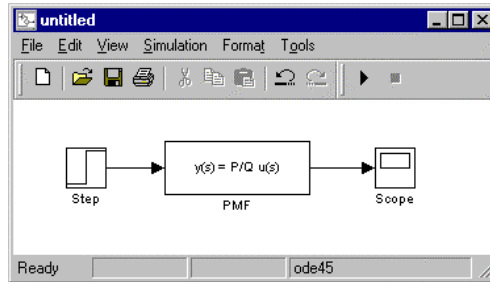


Fig. 10. A simple SIMULINK block diagram

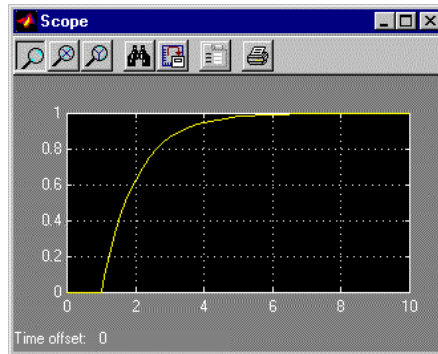


Fig. 11. Simulated step response

Algorithm	The macro is based on the SIMULINK's built-in blocks State space (Linear library) and Discrete state space (Discrete library) and uses the masking ability of SIMULINK along with the Polynomial Toolbox functions lmf2ss and rmf2ss for the transformation of the polynomial description of a system into state space form.	
Diagnostics	The macro displays no error messages when creating the library. If inconsistent data (such as a non-proper polynomial matrix fraction) are entered, however, then a standard error message may appear during the internal transformation into the state space model.	
See also	lmf2ss , rmf2ss	conversion of a left or right polynomial matrix fraction to a state space representation

polfit

Purpose Fit polynomial matrix to data

Syntax

```
[P,S] = polfit(pnts,Y)
[P,S] = polfit(pnts,Y,N)
[P,S] = polfit(pnts,Y,[],tol)
[P,S] = polfit(pnts,Y,N,tol)
```

Description Given a set of complex numbers **pnts**, a three-dimensional array **Y** such that $Y(:, :, i)$ represents a polynomial matrix evaluated at $\text{pnts}(i)$, and a matrix of degrees **N**, the command

```
P = polfit(pnts,Y,N)
```

finds a polynomial matrix **P** whose entries have degrees less than or equal to the corresponding entry of **N** such that

$$P(\text{pnts}(i)) \approx Y(:, :, i)$$

in a least-squares sense. **N** may also be a scalar in which case it applies to each entry of **P**. If the argument **N** is missing then it is set equal to `length(x)-1`. The command

```
[P,S] = polfit(pnts,Y,N)
```

returns the polynomial matrix P and a structure array S for use with `polyval` to obtain error estimates on predictions. See the MATLAB command `polyfit` for details.

The commands

```
polfit(pnts,Y,N,tol)
```

```
polfit(pnts,Y,[],tol)
```

work with zeroing specified by the input tolerance `tol`.

The length of `pnts` and the third size of Y must agree. The first and second sizes of Y must equal the size of N , unless N is a scalar.

Examples

Suppose that

```
pnts = 0:1:4;
```

and

```
Y
```

```
Y(:, :, 1) =
```

```
1      0      0
```

```
Y(:, :, 2) =
```

```
1      1      1
```

```

Y(:, :, 3) =
    1     2     4
Y(:, :, 4) =
    1     3     9
Y(:, :, 5) =
    1     4    16

```

Then

```
P = polfit(pnts,Y)
```

returns

```

P =
    1     s    s^2

```

Algorithm

The macro is based on the standard MATLAB routine `polyfit`.

Diagnostics

The macro issues error messages under the following conditions:

- There are too few or too many input arguments
- The first argument is not a vector
- The first or second argument is not a numeric array

- The degree matrix is not an integer array
- The sizes of the data sets do not match
- The size of the degree matrix does not match that of the data array

See also

`polyval` evaluate a polynomial matrix at a given set of points

polpart

Purpose Polynomial matrix part extraction

Syntax

`BH = polpart(B)`

`BH = polpart(B,n)`

`[BL BR] = polpart(B)`

`[BL BR] = polpart(B,n)`

Description

If B is a continuous-time para-Hermitian polynomial matrix, that is, a polynomial matrix in the variables s or p such that $B = B'$, then the command

`BH = polpart(B)`

extracts a polynomial matrix BH such that

$$B = BH' + BH$$

If B is a polynomial matrix in the variable z then it is considered to represent the discrete-time para-Hermitian (no longer polynomial) matrix $P(z) = z^{-n} B(z)$, where n is the “degree.” Thus, the discrete-time para-Hermitian polynomial matrix

$$P(z) = \begin{bmatrix} z+1/z & 2 \\ 2 & 3z^2 + 3/z^2 \end{bmatrix}$$

for instance, is represented by the polynomial matrix

$$B(z) = z^2 P(z) = \begin{bmatrix} z^3 + z^2 & 2z^2 \\ 2z^2 & 3z^4 + 3 \end{bmatrix}$$

where $n = 2$.

In the discrete-time case the command

BH = polpart(B,n)

yields a polynomial matrix **BH** such that

$$P(z) = BH(z) + BH^T(1/z)$$

Equivalently,

B = BH' + shift(BH,n)

Note that the prime denotes adjugation of discrete-time polynomial matrices in the sense of the Polynomial Toolbox.

If n is not specified then it is taken equal to **floor(deg(B)/2)**.

If the variable of B is q , d or z^{-1} then the command works similarly.

If B does not represent a para-Hermitian matrix then the command

[BL BR] = polpart(B,n)

extracts left and right parts of the square discrete-time polynomial matrix B such that

$$B = BL' + \text{shift}(BR, n)$$

Examples

Consider the continuous-time para-Hermitian polynomial matrix

$$B = \begin{bmatrix} 1+s^2 & 2*s \\ -2*s & 4 \end{bmatrix};$$

The command

$$BH = \text{polpart}(B)$$

returns

$$BH = \begin{bmatrix} 0.5 + 0.5s^2 & 0 \\ -2s & 2 \end{bmatrix}$$

We see that $B = BH + BH'$.

Next, consider the discrete-time para-Hermitian matrix

$$P(z) = \begin{bmatrix} z+1/z & 2 \\ 2 & 3z^2 + 3/z^2 \end{bmatrix}$$

This is represented by the polynomial matrix

$$B(z) = z^2 P(z) = \begin{bmatrix} z^3 + z & 2z^2 \\ 2z^2 & 3z^4 + 3 \end{bmatrix}$$

where the “degree” is $n = 2$.

Letting

```
B = [ z^3+z  2*z^2
      2*z^2  3*z^4+3 ];
```

the command

```
BH = polpart(B)
```

returns

```
BH =
      z      1
      1      3z^2
```

Indeed, $P(z) = BH(z) + BH^T(1/z)$.

If we modify B according to

```
B(1,1) = z^3
B =
```

$$\begin{array}{cc} z^3 & 2z^2 \\ 2z^2 & 3 + 3z^4 \end{array}$$

then it represents the matrix

$$P(z) = z^{-2}B(z) = \begin{bmatrix} z & 2 \\ 2 & 3z^2 + 3/z^2 \end{bmatrix}$$

This matrix is no longer para-Hermitian. The command

```
[BL BR] = polpart(B)
```

yields

$$\begin{array}{cc} \text{BL} = & \\ & 0 \quad 1 \\ & 1 \quad 3z^2 \\ \\ \text{BR} = & \\ & z \quad 1 \\ & 1 \quad 3z^2 \end{array}$$

Indeed,

$$BL^T(1/z) + BR(z) = \begin{bmatrix} 0 & 1 \\ 1 & 3/z^2 \end{bmatrix} + \begin{bmatrix} z & 1 \\ 1 & 3z^2 \end{bmatrix}$$

equals P .

Algorithm

The macro uses basic MATLAB and Polynomial Toolbox operations.

Diagnostics

The macro issues error messages under the following conditions:

- The input polynomial matrix is not square
- Two output arguments are encountered for a continuous-time input matrix
- The degree offset is invalid

A warning is issued if the polynomial matrix is not para-Hermitian when it is expected to be.

See also

ctranspose (')	conjugate transpose of a polynomial matrix
axxab , xaaxb	solution of symmetric linear polynomial matrix equations
spf	spectral factorization of symmetric polynomial matrices

polyder

Purpose Differentiate polynomial matrices

Syntax `Q = polyder(P)`

`Q = polyder(P,n)`

Description The command

`Q = polyder(P)`

computes the first derivative of the polynomial matrix P , while

`Q = polyder(P,n)`

computes its n th derivative.

Examples Consider

```
P = [ 1+s  s^2
      3*s  4  ];
```

We obtain successively

```
P1 = polyder(P)
```

```
P1 =
```

```

1      2s
3      0

P2 = polyder(P,2)

Constant polynomial matrix: 2-by-2

P2 =

    0      2
    0      0

P3 = polyder(P,3)

Zero polynomial matrix: 2-by-2, degree: -Inf

P3 =

    0      0
    0      0

```

Algorithm The macro uses standard MATLAB and Polynomial Toolbox operations.

Diagnostics An error message follows if an invalid second input argument is encountered.

polyval

Purpose Evaluate a polynomial matrix at a given set of points

Syntax

```
ev = polyval(P,pnt)

[ev,delta] = polyval(P,pnt,S)

ev = polyval(P,pnt,[],tol)

[ev,delta] = polyval(P,pnt,S,tol)
```

Description If P is a polynomial matrix and \mathbf{pnt} a given scalar then

```
ev = polyval(P,pnt)
```

evaluates the polynomial matrix P in the complex point \mathbf{pnt} . If \mathbf{pnt} is a vector of points $\mathbf{pnt} = [\mathbf{pnt1} \ \mathbf{pnt1} \ \dots \ \mathbf{pntn}]$ then \mathbf{ev} is the 3-dimensional array of values

```
ev(:, :, i) = P(pnti)
```

The command

```
[ev,delta] = polyval(P,pnt,S)
```

uses the output structure array S generated by `polfit` to obtain error estimates \mathbf{delta} on predictions. If \mathbf{pnt} is a vector then \mathbf{delta} is a 3-dimensional array.

The commands

```
ev = polyval(P,pnt,s,tol)
```

```
ev = polyval(P,pnt,[],tol)
```

evaluate the polynomial matrix P in the vector `pnt` and set to zero all entries in the result whose magnitude is less than `tol`.

Examples

Consider the polynomial matrix

```
P = [ 1  s  s^2 ];
```

and let `pnt` be the array

```
pnt = 0:1:4;
```

Then

```
ev = polyval(P,pnt)
```

returns

```
ev(:, :, 1) =
```

```
1      0      0
```

```
ev(:, :, 2) =
```

```
1      1      1
```

```

ev(:,:,3) =
    1     2     4
ev(:,:,4) =
    1     3     9
ev(:,:,5) =
    1     4    16

```

For an application involving the structure array *S* see the description of the routine `polfit`.

Algorithm

The macro uses the standard MATLAB routine `polyval`.

Diagnostics

The macro issues error messages if the set of evaluation points is empty or is not a vector.

See also

`polfit` fit a polynomial matrix to data

pplace

Purpose

Pole placement by polynomial methods

Syntax

```
[Nc,Dc] = pplace(N,D,poles), [Nc,Dc]=pplace(N,D,poles,'l')
```

```
[Nc,Dc] = pplace(N,D,poles,'r')
```

```
[Nc,Dc,E,F,degT] =  
pplace(N,D,poles), [Nc,Dc]=pplace(N,D,poles,'l')
```

```
[Nc,Dc,E,F,degT] = pplace(N,D,poles,'r')
```

```
[Nc,Dc] = pplace(N,D,R)
```

```
[Nc,Dc] = pplace(N,D,R,'r')
```

Description

Given a linear time-invariant plant transfer matrix

$$P(v) = D^{-1}(v)N(v)$$

where v can be any of s, p, z, q, z^{-1} and d , and a vector of desired closed-loop pole locations `poles`, the commands

```
[Nc,Dc] = pplace(N,D,poles)
```

```
[Nc,Dc] = pplace(N,D,poles,'l')
```

compute a controller with transfer matrix

$$Q(v) = N_C(v)D_C^{-1}(v)$$

such that the feedback shown in Fig. 12 places the closed-loop poles at the desired locations `poles`. The multiplicity of the poles is increased if necessary. The resulting system may have real or complex coefficients depending on whether or not the desired poles are self-conjugate.

For the same plant and the same desired locations vector, the command

```
[Nc,Dc,E,F,degT]=pplace(N,D,poles)
```

may be used to get the parameterization

$$Q(v) = (N_C(v) + E(v)T(v))(D_C(v) - F(v)T(v))^{-1}$$

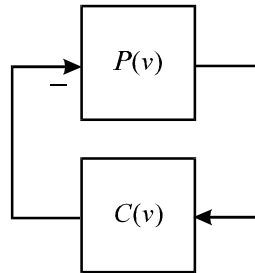


Fig. 12. Feedback structure

of all other controllers yielding the same dynamics. $T(v)$ is an arbitrary polynomial matrix parameter of compatible size and of degree bounded by `degT`.

Note: The macro is intended for single-input single-output plants. It does its job for multi-input multi-output systems as well but the user should be aware of the fact that just assigning pole locations need not be enough. In the multi-input multi-output case, the desired behavior typically depends also on the closed-loop invariant polynomials rather than on the pole locations only. In fact, the assignment of invariant polynomials is very easy. All it needs is to put the desired invariant polynomials p_i into a diagonal matrix $R(v)$ of the same size as $D(v)$ and call

```
[Nc,Dc] = pplace(N,D,R)
```

Dually, if the plant transfer matrix is given by

$$P(v) = N(v)D^{-1}(v)$$

then the command

```
[Nc,Dc] = pplace(N,D,poles,'r')
```

computes a desired controller transfer matrix in the form

$$Q(v) = D_C^{-1}(v)N_C(v)$$

while the command

```
[Nc,Dc,E,F,degT] = pplace(N,D,poles,'r')
```

gives rise to the parametrization

$$Q(v) = (D_c(v) - T(v)F(v))^{-1}(N_c(v) + T(v)E(v))$$

$T(v)$ is an arbitrary polynomial matrix parameter of compatible size and degree up to `degT`.

To prescribe not only the locations of the poles but also their distribution among the invariant polynomials p_i put them into a diagonal matrix $R(v)$ of the same size as $D(v)$ and call

```
[Nc,Dc] = pplace(N,D,R,'r').
```

Note: If the plant is strictly proper (or strictly causal) then the resulting controller is always proper (causal). Otherwise its properness (causality) is not guaranteed and should be checked separately.

Noce: The macro handles only coprime fraction descriptions of the plant. If the fraction has a common factor then the user should cancel it before running `pplace`.

Example

Consider a simple continuous-time plant described by

```
d=2-3*s+s^2,n=s+1
```

```
d =
```

```
2 - 3s + s^2
```

```
n =
      1 + s
```

that has two (unstable) poles

```
roots(d)
ans =
      2.0000
      1.0000
```

The poles may be shifted arbitrarily with a first order controller. The resulting feedback system has three poles. Let us place them to the locations $s_1 = -1$, $s_2 = -1+j$ and $s_3 = -1-j$. A corresponding controller results from

```
[nc,dc]=pplace(n,d,[-1,-1+j,-1-j])
nc =
      5s
dc =
      1 + s
```

and has the transfer function

$$\frac{n_c(s)}{d_c(s)} = \frac{5s}{1+s}.$$

Indeed,

```
r=d*dc+n*nc
r =
      2 + 4s + 3s^2 + s^3
```

and

```
roots(r)
ans =
-1.0000 + 1.0000i
-1.0000 - 1.0000i
-1.0000
```

as desired. There are no other proper controllers placing poles exactly that way as

```
[nc,dc,f,g,degT]=ppplace(n,d,[-1,-1+j,-1-j])
nc =
      5s
```



```

dc =
      1 + s

f =
      2 - 3s + s^2

g =
      1 + s

degT =
      -Inf

```

There is no degree of freedom offered by the parameter $T(s) = 0$ in this example.

Algorithm

Given $P = D^{-1}N$ the macro computes $C = N_C D_C^{-1}$ by solving the linear polynomial matrix equation $DD_C + NN_C = R$ for a suitable right hand side matrix R . Similarly, given $P = ND^{-1}$ the controller $C = D_C^{-1}N_C$ is constructed with the help of $D_C D + N_C N = R$ (Kucera, 1979; Kucera, 1991).

Diagnostics

The macro returns error messages in the following situations:

- The input arguments are not polynomial objects
- The input matrices have inconsistent dimensions
- The input string is incorrect

- The input polynomial matrices are not coprime

See also

stab stabilization

debe deadbeat regulator design

pplot, pplot3

Purpose Polynomial two- or three-dimensional matrix graphs

Syntax

```
pplot(P)
```

```
pplot(P,n)
```

```
pplot(P,'last')
```

```
pplot3(P)
```

```
pplot3(P,n)
```

```
pplot3(P,'last')
```

Description

The command

```
pplot(P)
```

creates a new figure with a 2-dimensional plot of the polynomial matrix P , while

```
pplot(P,n)
```

creates a 2-dimensional plot and puts it into an existing figure window with number n . The command

```
pplot(P,'last')
```

puts it into the last created or mouse-activated figure window.

The color of each cell corresponds to the degree of the corresponding entry of P . Empty white places correspond to zero entries. The color map may be changed by the menu “Color.”

The commands

```
pplot3(...)
```

create 3-dimensional plots. The height of each bar corresponds to the degree of the corresponding entry of P . The color of each bar corresponds to the magnitude of the leading coefficient of the corresponding entry of P . Empty white places correspond to zero entries. The color map may be changed by the menu “Color.” The button “Rotate” turns a mouse-based interactive rotation of the plot view on or off.

Example

The command

```
P = prand(floor(5*rand(3,3)),5,5,'int')
```

generates the unattractive polynomial matrix

```
Polynomial matrix in s: 5-by-5, degree: 4
```

```
P =
```

```
Columns 1 through 2
```

```
-1 - s + s^2
```

```
-4 - s - 5s^3 + 5s^4
```

Commands — Online reference

```

6                                -3 + 4s + s^2 - 2s^3 - 4s^4
7 - 4s + 4s^2 - 3s^3           4 + 4s - 4s^2 + 3s^3
-10 + s + 7s^2 + s^4           s + 4s^2 + s^3 + 4s^4
-3 + s - 3s^2 + 2s^4           6 + 4s - 13s^2 + s^3 + 4s^4

Columns 3 through 4

-6 - 4s - 6s^2                 5 - 3s - 9s^2 + 4s^3 +
7s^4

1                                -9 - 8s + 10s^2 + 13s^4
5 - 5s + 2s^2                 -6 - s - s^2 - 5s^3 -
10s^4

3 + s + s^2 - 5s^3 - 5s^4      2s - 4s^2 + 4s^3 + s^4
4s + 4s^2 + 7s^3 + 6s^4      5 - 8s - 5s^2 + 3s^3 -
5s^4

Column 5

3 + s - 4s^2 - 4s^3 + 6s^4
-1 + s + 5s^2 - s^3 - 3s^4
-5 - 9s^2 + 6s^3 - 3s^4

```

$$14 + 2s - 7s^2 + 2s^3 + 2s^4$$

$$-3 + 3s + 5s^3 - 7s^4$$

The degrees of its entries are

```
deg(P, 'ent')
```

```
ans =
```

2	4	2	4	4
0	4	0	4	4
3	3	2	4	4
4	4	4	4	4
4	4	4	4	4

The two-dimensional plot of Fig. 13 is produced by the command

```
pplot(P)
```

It confirms that the low-degree entries are all in the upper left half corner.

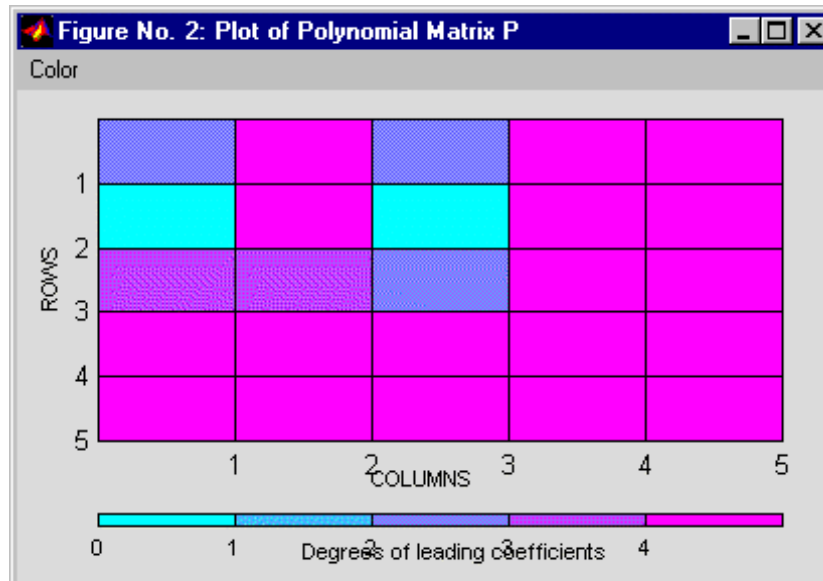


Fig. 13. 2-D plot

Typing

```
pplot3(P)
```

generates the three-dimensional plot of Fig. 14 that unequivocally singles out the entry with the largest leading coefficient.

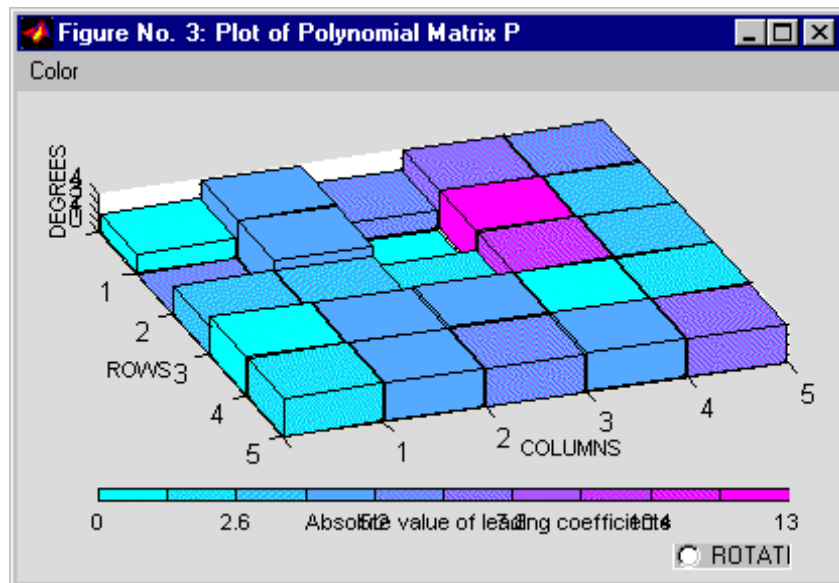


Fig. 14. 3-D plot

Rotating the plot results in the view of Fig. 15.

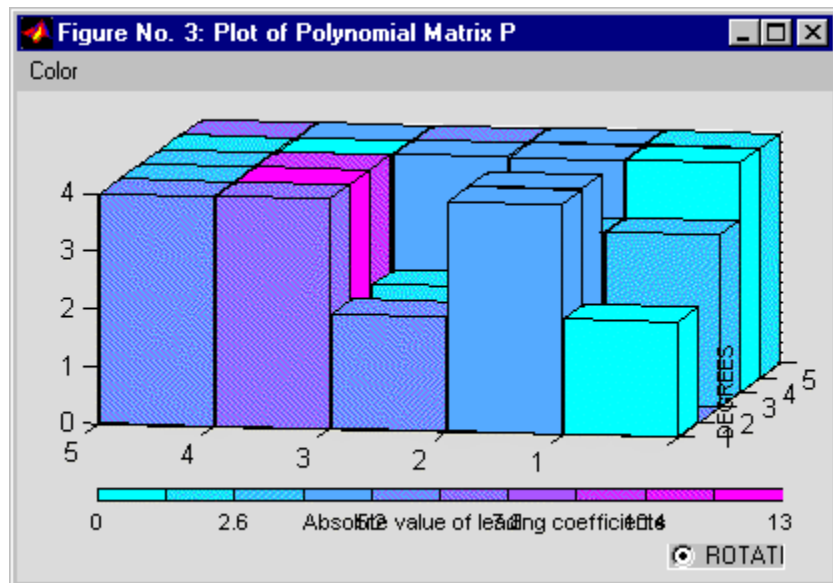


Fig. 15. Rotated 3-D plot

Algorithm

The macros rely on standard Polynomial Toolbox and MATLAB 5 commands.

Diagnostics

The macros display error messages if they do not have enough input arguments or the input is not a polynomial matrix.

See also

`pme` polynomial matrix editor

pprop, symbol, userdata

Purpose Display or set the properties of a polynomial object

 Return or set the variable symbol of a polynomial object

 Set or return the user data of a polynomial object

Syntax `pprop(P)`

`pprop(P,value)`

`pprop(P,value1,value2,...)`

`sP = symbol(P)`

`symbol(P,string)`

`userdata(P,argm)`

`u = userdata(P)`

Description The command

`pprop(P)`

 displays all the properties of P and their admissible values. The command

`pprop(P,value)`

sets the property of the polynomial matrix P corresponding to `value` equal to the value `value`, while

```
pprop(P,value1,value2,...)
```

sets multiple property values of P equal to the values `value1`, `value2`, ...

The command

```
sP = symbol(P)
```

returns the current value of the variable symbol of the polynomial matrix P , while

```
symbol(P,string)
```

sets the variable symbol of P equal to the value `string` (and returns this value).

The command

```
userdata(P,argm)
```

sets the user data of the polynomial object P equal to `argm`. The function call

```
u = userdata(P)
```

returns the current value of the user data of P .

Examples

After defining the polynomial matrix P as

```
P = [1+s 2 3];
```

the properties of the polynomial matrix P may be viewed by typing

```
pprop(P)
```

```
POLYNOMIAL MATRIX
```

```
size      1-by-3
```

```
degree    1
```

PROPERTY NAME:	CURRENT VALUE:	AVAILABLE VALUES:
variable symbol	s	's','p','z^-'
	1','d','z','q'	
user data	[]	arbitrary

We modify both available properties by typing

```
pprop(P,'z','Redefined matrix')
```

The effect may be inspected immediately

```
pprop(P)
```

```
POLYNOMIAL MATRIX
```

```
size      1-by-3
```

```
degree      1
```

PROPERTY NAME:	CURRENT VALUE:	AVAILABLE VALUES:
variable symbol	z	's','p','z'-
	1','d','z','q'	
user data	Redefined matrix	arbitrary

We may also observe the change when we type

```
P
P =
      1 + z      2      3
```

The command

```
symbol(P)
ans =
z
```

confirms once again that P is polynomial in the symbol z . We change the symbol back to s with the command

```
symbol(P,'s'); P
```

and indeed

```
P =
      1 + s      2      3
```

Typing

```
userdata(P)
```

returns

```
ans =
Redefined matrix
```

Algorithm

The macros use standard MATLAB 5 commands.

Diagnostics

The macros return error messages if the input is inappropriate.

See also

gprop set or modify the global default properties of polynomial matrices
gensym set the global default variable

prand

Purpose Generate a polynomial matrix with random coefficients

Syntax

```
P = prand(degP,n,m)
P = prand(degP,n)
P = prand(degP)
P = prand
P = prand(degP,n,m,options)
```

Description The command

```
P = prand(degP,n,m)
```

generates a “random” $n \times m$ polynomial matrix P of degree `degP` with normally distributed coefficients.

If m is missing then a square $n \times n$ matrix is created. If also n is missing then the sizes of `degP` are used.

If `degP` is a matrix of integers then its entries specify the degrees of the corresponding entries of P . If `degP` is a column or row vector then its entries specify the row or column degrees of P , respectively.

To randomize also the degrees of the entries, rows or columns within the interval $[0, \text{degP}]$ the options `'ent'`, `'row'` or `'col'`, respectively, may be included.

An input string `'uni'` instead of these strings makes P a square $n \times n$ unimodular matrix such that $\det(P) = 1$.

The option `'sta'` makes P a “stable” polynomial matrix. In this case `degP` specifies the number of roots. This count includes possible multiplicities. What stability means depends on the default global variable — see `isstable`.

Any of the syntaxes described may be combined with the string `'int'` to produce “small integer” coefficients.

Examples

The command

```
P = prand(2,2,2)
```

results in

```
P =
```

```

-1.7 + 0.13s + 0.29s^2    -1.1 + 1.2s + 1.2s^2
-0.038 + 0.33s + 0.17s^2  -0.19 + 0.73s - 0.59s^2

```

while

```
P = prand(2,2,2,'int')
```


produces

```
P =
```

$$\begin{bmatrix} -1 + s + 5s^2 & -4s^2 \\ 1 - 7s + 4s^2 & 8 - 3s + 4s^2 \end{bmatrix}$$

A random matrix of which the first row has degree 1 and the second has degree 2 follows by the command

```
P = prand([1;2],2,2)
```

```
P =
```

$$\begin{bmatrix} -1.6 - 1.4s & 0.57 - 0.4s \\ 0.69 + 0.82s + 0.71s^2 & 1.3 + 0.67s + 1.2s^2 \end{bmatrix}$$

Typing `prand` without further argument or `prand('int')` generates unpredictable polynomial matrices as intended.

It may be checked that

```
P = prand(2,3,'int','uni')
```

```
P =
```

$$\begin{bmatrix} -2 & 2 - 2s & 1 \\ -3 + s + s^2 & 2 - 3s + 2s^2 & 1 - s \end{bmatrix}$$

$$\begin{array}{ccc} -1 & - & s \\ 1 & - & 2s \\ 1 & & \end{array}$$

is unimodular by typing

```
det(P)
```

```
Constant polynomial matrix: 1-by-1
```

```
ans =
```

```
1
```

The command

```
gprop s, P = prand(2,3,3,'sta','int')
```

produces the polynomial matrix

```
P =
```

$$\begin{array}{ccc} 9 + 19s + 2s^2 & -9s - s^2 & -18 - 29s - 3s^2 \\ -8 - 2s & 4 + s & -4 + 3s + s^2 \\ 0 & 0 & 1 \end{array}$$

P has two roots

```
roots(P)
```

```
ans =
```

```
-9.0000
```

```
-4.0000
```

both with negative real parts. On the other hand

```
gprop z, P = prand(2,3,3,'sta','int')
```

generates a matrix

```
Polynomial matrix in z: 3-by-3, degree: 2
```

```
P =
```

```
Columns 1 through 2
```

```
2.8e+002 + 4z - 2e+002z^2    -28
```

```
-5.6e+002 - 6.3e+002z      2e+002 + 2e+002z
```

```
-1                            0
```

```
Column 3
```

```
-4.5e+002 - 2e+002z + 2e+002z^2
```

```
3.4e+002 + 4.3e+002z
```

```
1
```

The two roots

```

rts = roots(P), abs(rts)

rts =

    -0.9800 + 0.1400i
    -0.9800 - 0.1400i

ans =

    0.9899
    0.9899

```

both have magnitude less than 1.

Algorithm

The macros mostly use standard MATLAB and Polynomial Toolbox operations.

Random unimodular matrices are generated as the product of a lower triangular unimodular matrix and an upper triangular unimodular matrix. Each triangular unimodular matrix has ones on the diagonal and random polynomials in the nonzero off-diagonal entries.

A random matrix with prescribed roots may be generated as the product USV with U and V random unimodular and S diagonal with polynomials of degree 1 or 2 on the diagonal. The roots may easily be randomly placed in the left-half complex plane with integer real and imaginary parts if desired. First or second-order polynomials with integral coefficients and roots inside the unit circle follow from

polynomials with integral coefficients and roots in the left-half plane by the substitution $s = (z - 1) / (z + 1)$.

Diagnostics

An error message follows if any of several inconsistencies is encountered in the input.

See also

`mono` create monomial vector or matrix

prod, sum, trace

Purpose Product or sum of the entries of a polynomial matrix

Trace of a polynomial matrix

Syntax

`q = prod(P[,tol])`

`q = prod(P,dim[,tol])`

`q = sum(P[,tol])`

`q = sum(P,dim[,tol])`

`t = trace(P[,tol])`

Description

If P is a polynomial column or row vector then

`q = prod(P)`

returns the product of its entries. If P is a matrix then `q = prod(P)` treats the columns of P as vectors, returning a row vector of the products of each column. The command

`q = prod(P,dim)`

takes the products along the dimension specified by the integer `dim`.

The optional input argument `tol` specifies the zeroing tolerance. Its default value is the global zeroing tolerance.

Similarly, if P is a polynomial column or row vector then

```
q = sum(P)
```

returns the sum of its entries. If P is a matrix then `q = sum(P)` treats the columns of P as vectors, returning a row vector of the sums of each column. The command

```
q = sum(P,dim)
```

takes the sums along the dimension specified by the integer `dim`.

The optional input argument `tol` specifies the zeroing tolerance. Its default value is the global zeroing tolerance.

The command

```
t = trace(P)
```

computes the sum of the diagonal entries of P with zeroing activated through the global zeroing variable. The optional input argument `tol` specifies the zeroing tolerance.

Examples

Let

```
P = [ 1+s  2*s^2  3+4*s  
      5      6      7      ];
```

Then

```
prod(P)
```

results in

```
ans =
```

```
5 + 5s    12s^2    21 + 28s
```

while

```
trace(P)
```

yields

```
ans =
```

```
7 + s
```

Algorithm

The macros rely on standard MATLAB 5 and Polynomial Toolbox operations.

Diagnostics

The macros `prod` and `sum` display an error message if an unknown command option is encountered.

ptopex

Purpose Enumeration of extreme polynomials for a polytopic family

Syntax `[pex1,pex2,...,pexm] = ptopex(p0,p1,p2,...,pn,Qbounds)`

`EX = ptopex(p0,p1,p2,...,pn,Qbounds)`

Description A polytopic family of polynomials $P = \{p(\cdot, q) : q \in Q\}$ (Barmish, 1996) is described by

$$p(s, q) = p_0(s) + \sum_{i=1}^l q_i p_i(s)$$

where $p_0(s), p_1(s), \dots, p_n(s)$ are polynomials and $q_i \in [q_i^-, q_i^+]$, $i = 1, 2, \dots, l$, are uncertain parameters from a parameter bounding set $Q = \text{conv}\{q^k\}$, which is generated by its $m = 2^l$ extreme points $q = (q_1^-, q_2^-, \dots, q_n^-)$. We denote the bounds of Q by

$$Q_{\text{bounds}} = \begin{bmatrix} q_1^- & q_1^+ \\ q_2^- & q_2^+ \\ \dots & \dots \\ q_n^- & q_n^+ \end{bmatrix}$$

The command

```
[pex1,pex2,...,pexm] = ptopex(p0,p1,p2,...,pn,Qbounds)
```

computes the $m = 2^n$ extreme polynomials $p(s, q^j)$.

If only one output argument is used then the command

```
EX = ptopex(p0,p1,p2,...,pn,Qbounds)
```

returns all the extreme polynomials in the polynomial column vector **EX**.

Extreme polynomials generate the polytopic family so that $P = \text{conv}\{p(s, q^j)\}$, include its extreme points and may be used for its robust stability analysis.

Example

Let a polytope of polynomials be described by

$$p(s, q) = (2q_1 - q_2 + 5) + (4q_1 + 3q_2 + 2)s + s^2,$$

Equivalently,

$$p(s, q) = (5 + 2s + s^2) + q_1(2 + 4s) + q_2(-1 + 3s)$$

and

$$|q_1| \leq 1, \quad |q_2| \leq 1.$$

To input the data, type

```
p0 = 5+2*s+s^2, p1 = 2+4*s, p2=-1+3*s, Qbounds = [-1 1;-1 1];
```

The uncertainty bounding set has four extremes $q^1 = (-1, -1)$, $q^2 = (1, -1)$, $q^3 = (-1, 1)$ and $q^4 = (1, 1)$. The four associated extreme polynomials (generators) of the polynomial family are

$$\begin{aligned} p(s, q^1) &= 4 - 5s + s^2 \\ p(s, q^2) &= 8 + 3s + s^2 \\ p(s, q^3) &= 2 + s + s^2 \\ p(s, q^4) &= 6 + 9s + s^2 \end{aligned}$$

They are computed by

```
EX = ptopex(p0,p1,p2,Qbounds)
```

```
EX =
```

```
4 - 5s + s^2
```

```
8 + 3s + s^2
```

```
2 + s + s^2
```

```
6 + 9s + s^2
```

Algorithm

The routine uses standard routines from the Polynomial Toolbox.

Diagnostics

The macro returns error messages in the following situations:

- The input arguments are not polynomial objects

- The input matrices have inconsistent dimensions

See also

`ptopplot` plot polytopic value set

ptopplot

Purpose	Plot the value set for a polytopic family of uncertain polynomials
Syntax	<code>ptopplot(p0,p1,p2,...,pn,Qbounds,freq)</code> <code>ptopplot(p0,p1,p2,...,pn,Qbounds,freq,'new')</code>
Description	A polytopic family of polynomials $P = \{p(\cdot, q) : q \in Q\}$ is described by $p(s, q) = p_0(s) + \sum_{i=1}^l q_i p_i(s)$ where $p_0(s), p_1(s), \dots, p_n(s)$ are polynomials and $q_i \in [q_i^-, q_i^+]$ are uncertain parameters. The parameter bounding set Q is a box given by

$$Q_{\text{bounds}} = \begin{bmatrix} q_1^- & q_1^+ \\ q_2^- & q_2^+ \\ \dots & \dots \\ q_n^- & q_n^+ \end{bmatrix}.$$

The set Q is generated by its extreme points $q^K = (q_1^{\text{sg}_1}, q_2^{\text{sg}_2}, \dots, q_n^{\text{sg}_n})$, where each sg_i is either $-$ or $+$, and $Q = \text{conv}\{q^K\}$.

For a fixed generalized frequency $z \in \mathbf{C}$, we define the *value set* $p(z, Q)$ (Barmish, 1996; Bose and Zeheb, 1986) as the set of all possible complex values that the

$p(z, q)$ assumes when q ranges over Q . If $p(z, q)$ is a polytop of polynomials then the value set $p(z, Q)$ is a polygon in the complex plane generated by $\{p(z, q^j)\}$, that is,

$$p(z, Q) = \text{conv}\{p(z, q^j)\}.$$

The command

```
ptopplot(p0,p1,p2,...,pn,Qbounds,freq)
```

plots the polygonal value set $p(z, Q)$ for the given fixed complex frequency z given by **freq**. If **freq** is a vector of frequencies then the command

```
ptopplot(p0,p1,p2,...,pn,Qbounds,freq)
```

plots the polygonal value set “in motion.” This may be used a simple graphical check for robust stability of $p(s, q)$ based on the Zero Exclusion Condition.

The command

```
ptopplot(p0,p1,p2,...,pn,Qbounds,freq, 'new')
```

opens a new figure window for the plot leaving existing windows unchanged.

Note: The macro plots polygonal value sets $p(z, Q)$ of a proper shape for *generalized* (possibly complex) *frequencies* z given by the input argument **freq**. Owing to this feature, it may be used to test robust stability for various stability regions including the unit disc. For standard (continuous-time) left half-plane

stability, however, one must input a complex `freq = z = jw`, instead of just a real `w` such as employed in half-plane stability oriented macros (`khplot`).

Note: The macro assumes that each uncertain parameter q_i ranges over a straight-line segment connecting the points q_i^- and q_i^+ even if q_i is complex. If a complex interval $[r^-, r^+] + j[p^-, p^+]$ is required then it must be described by two parameters: a real $q_r \in [r^-, r^+]$ and a purely imaginary $q_p \in [p^-, p^+]$.

Example 1

Consider the uncertain polynomial (Barmish 1994, p. 143)

$$p(s, q) = (3 + 3s + 3s^2 + s^3) + q_1(1 + 3s + 3s^2 + 2s^3) \\ + q_2(-1 + s - 3s^2 - s^3) + q_3(2 + s + s^2 + 2s^3)$$

with $|q_i| \leq 0.245$. After entering the data

```
p0 = pol([3 3 3 1],3); p1 = pol([1 3 3 2],3);
p2 = pol([-1 1 -3 -1],3); p3 = pol([2 1 1 2],3);
q = [-0.245 0.245; -0.245 0.245; -0.245 0.245];
```

we may plot the value set of Fig. 16 for thirty evenly spaced frequencies between $w = 0$ and $w = 1.5$ by typing

```
ptopplot(p0,p1,p2,p3,q,j*(0:1.5/30:1.5))
```

As the family above has a stable member $p(s,0)$ and the Zero Exclusion Condition is evidently satisfied the family is robustly stable.

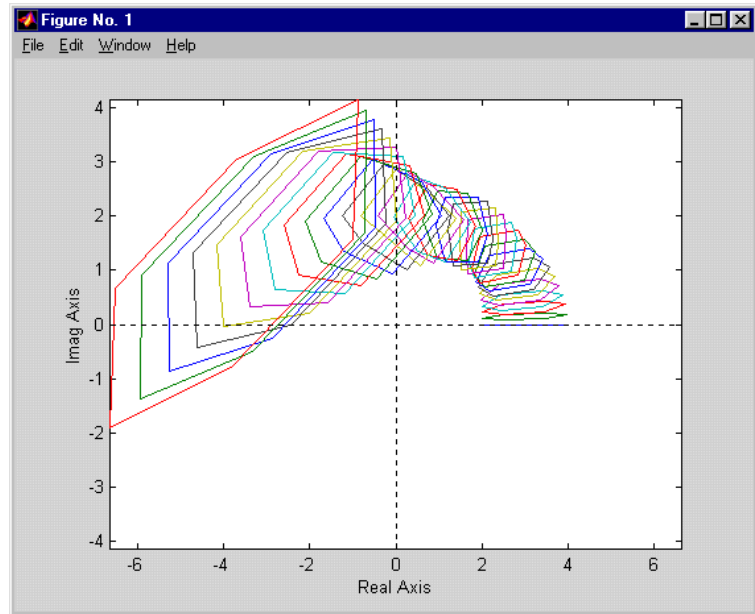


Fig. 16. Value set plot

Example 2

We check the robust Schur stability a polytopic family of the discrete-time polynomials described by

$$p(z, q) = z^3 + (0.5 + q)z^2 - (0.25 + q)$$

(Barmish, 1994, p. 115) or, equivalently, by

$$p(z, q) = (-0.25 + 0.5z^2 + z^3) + q(-1 + z^2)$$

where $Q = [-0.3, 0.3]$. We first input the data

```
p0 = -0.25+0.5*z^2+z^3, p1 = -1+z^2, Qbounds = [-0.3 0.3]
p0 =
    -0.25 + 0.5z^2 + z^3
p1 =
    -1 + z^2
Qbounds =
    -0.3000    0.3000
```

To apply the Zero Exclusion Condition we first identify one stable member. Indeed, by taking $q = q^0 = 0$ we have $p(z, q^0) = z^3 + 0.5z^2 - 0.25$, which is Schur stable:

```
roots(p0)
ans =
    -0.5000 + 0.5000i
    -0.5000 - 0.5000i
```

0.5000

Next we sweep z around the unit circle while plotting the value set $p(z, Q)$. In this particular example the value set is the straight-line segment joining $p(z, -0.3)$ and $p(z, 0.3)$. Fig. 17 shows it in motion using 1000 evaluation points evenly spaced around the unit circle as obtained by the command

```
ptopplot(p0,p1,Qbounds,exp(j*(0:2*pi/1000:2*pi)))
```

It is clear by inspection that $0 \notin p(z, Q)$ for all z . Hence, we conclude that $p(z, Q)$ is robustly stable.

Example 3

Finally, we check the robust Schur stability of the discrete-time interval polynomial

$$p(z, q) = -\frac{1}{3} + \frac{2}{3}z^2 + \left[-\frac{17}{8}, \frac{17}{8}\right]z^3 + z^4$$

First we express it in the polytopic form

$$p(z, q) = \left(-\frac{1}{3} + \frac{2}{3}z^2 - \frac{17}{8}z^3 + z^4\right) + qz^3$$

with

$$q \in \left[0, \frac{17}{4}\right].$$

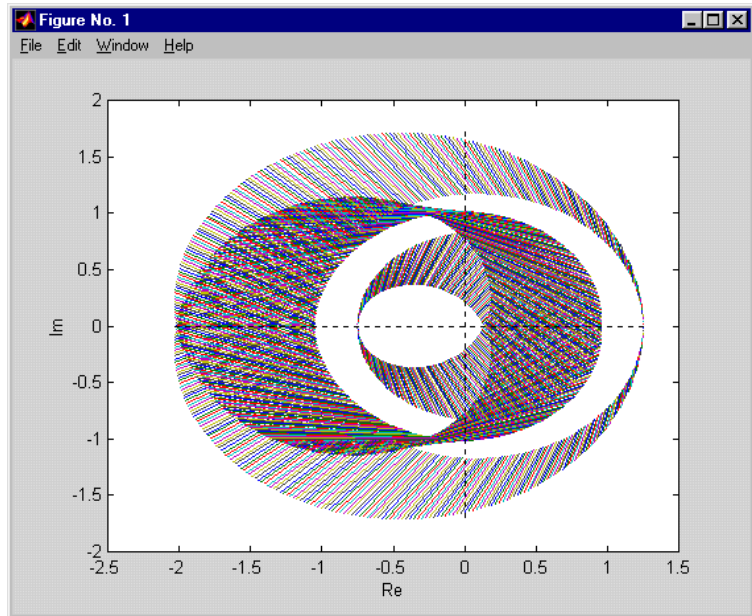


Fig. 17. Value set plot for Example 2

Next we enter the data

```
p0 = pol([-1/3 0 2/3 -17/8 1],4,'z'), p1 = z^3,
p0 =
```

```

-0.33 + 1.5z^2 - 2.1z^3 + z^4

p1 =

z^3

Qbounds = [0 17/4]

Qbounds =

0      4.2500

```

Now we plot in Fig. 18 the value set with z sweeping around the unit circle with the help of the command

```
ptopplot(p0,p1,Qbounds,exp(j*(0:0.01:1)*2*pi))
```

We see that the Zero Exclusion Condition is violated and, hence, the interval polynomial fails to be robustly Schur stable. This observation together with the fact that both extreme polynomials

```

[pmin,pmax] = ptopex(p0,p1,Qbounds)

pmin =

-0.33 + 1.5z^2 - 2.1z^3 + z^4

pmax =

-0.33 + 1.5z^2 + 2.1z^3 + z^4

```

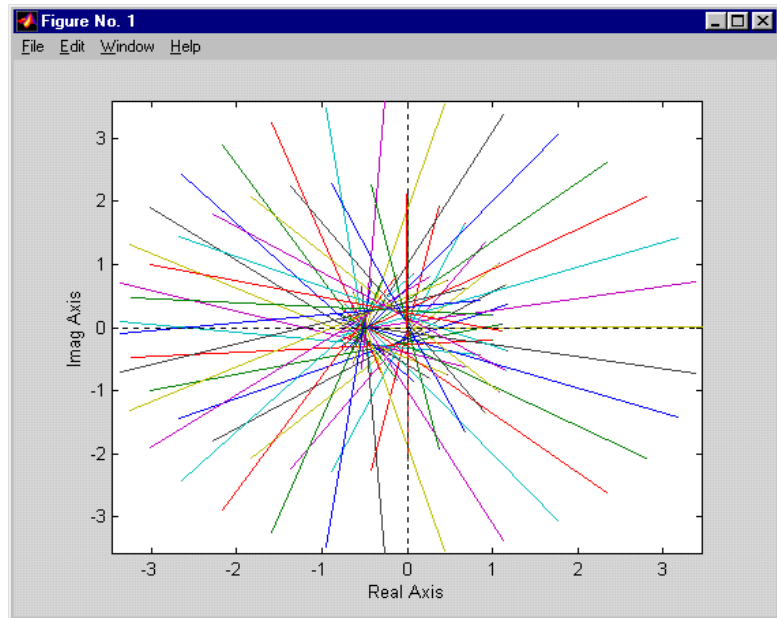


Fig. 18. Value set plot for Example 3

```
isstable(pmin), isstable(pmax)
```

```
ans =
```

```

1
ans =
1

```

are Schur stable indicates that Kharitonov-like extreme point results do not hold for Schur stability of polynomials with degree 4.

Algorithm

The macro first enumerates the extreme polynomials for $p(z, Q)$ using `ptopex`. Then it evaluates each of them at z via `polyval` and plots the convex hull of the resulting values to obtain the value set.

Diagnostics

The macro returns error messages in the following situations:

- The input arguments are not polynomial objects
- The input matrices have inconsistent dimensions
- The argument `freq` is either incorrect or missing.

See also

`ptopex` extreme polynomials for polytopic polynomial family
`khplot` plot Kharitonov rectangles

pzer

Purpose Perform zeroing on a polynomial matrix

Syntax `PZ = pzer(P)`

`PZ = pzer(P,ZEROING)`

Description The command

`PZ = pzer(P)`

sets the coefficients in P equal to zero whose absolute value is less than the current value of the global zeroing tolerance. The command

`PZ = pzer(P,ZEROING)`

uses the value of the argument `ZEROING` as the local tolerance.

To suppress zeroing globally set the global zeroing tolerance equal to zero using the command

`gprop 0`

Examples Typing

`P = [eps*s+1 eps+s]`

results in

```
P =  
1 + 2.2e-016s      2.2e-016 + s
```

In turn, typing

```
P = pzer(P)
```

modifies P to

```
P =  
1      s
```

Algorithm

The macro uses standard MATLAB 5 commands.

Diagnostics

The macro issues an error message if it finds too few input arguments.

See also

`plus`, `minus`, `times` arithmetic operations

qzord

Purpose Ordered QZ transformation

Syntax

```
[AA,BB,Q,Z] = qzord(A,B)

[AA,BB,Q,Z] = qzord(A,B,'partial')

[AA,BB,Q,Z] = qzord(A,B,'full')

[AA,BB,Q,Z] = qzord(A,B,'partial',tol)

[AA,BB,Q,Z] = qzord(A,B,'full',tol)
```

Description The command

```
[AA,BB,Q,Z] = qzord(A,B)
```

produces the ordered QZ transformation of the constant matrices A and B . **AA** and **BB** are upper triangular constant matrices and Q and Z nonsingular unitary matrices such that $QAZ = \mathbf{AA}$, $QBZ = \mathbf{BB}$. The generalized eigenvalues of the matrix pair (A, B) are the ratios of the diagonal entries of **AA** and **BB**.

If the commands

```
[AA,BB,Q,Z] = qzord(A,B)
```

```
[AA,BB,Q,Z] = qzord(A,B,'partial')
```

are used then the generalized eigenvalues are ordered such that the infinite generalized eigenvalues come last. If

```
[AA,BB,Q,Z] = qzord(A,B,'full')
```

is used then the generalized eigenvalues are ordered according to increasing real parts with the infinite generalized eigenvalues last.

The optional input argument `tol` sets the tolerance that is used to determine whether a generalized value is infinite.

Examples

Consider the matrices

```
A = [ 11    -7     2    12
      6     16     8    -7
      0      0   -15     5
      3     17    -8    -4 ];

B = [ -2     -1     -1     0
     -14     -8     5     0
     -13     -8     3     0
      9      8      7     0 ];

[AA,BB] = qzord(A,B)
```

```
AA =
    6.3213    20.7138     8.4444    -6.3634
   -0.0000     0.8513    -3.0038     0.3346
    0.0000    -0.0000    17.2160    -2.6845
    0.0000    -0.0000         0   -21.3711

BB =
    2.5471    -5.8621    -1.2364    -7.1106
         0   -22.4140     9.1457     6.7722
         0     0.0000    -0.8909    -1.1712
         0     0.0000    -0.0000    -0.0000
```

We compute the generalized eigenvalues by typing

```
d = diag(AA)./diag(BB); d(1:3)
ans =
    2.4817
   -0.0380
  -19.3241
```

There is a single infinite generalized eigenvalue, which comes last. The remaining eigenvalues are not ordered, at least not according to increasing real parts. To order them we use the command

```
[AA,BB] = qzord(A,B,'full');  
  
d = diag(AA)./diag(BB); d(1:3)  
  
ans =  
  
-19.3241  
  
-0.0380  
  
2.4817
```

The finite eigenvalues are now ordered according to increasing real part.

Algorithm

First the QZ algorithm is used to transform A and B to upper triangular form \mathbf{AA} and \mathbf{BB} (see `qz`). Adjacent diagonal entries of \mathbf{AA} and \mathbf{BB} may simultaneously be transformed using Givens rotations (see `cgivens1`) from both the left and the right such that the corresponding generalized eigenvalues are interchanged. By suitable interchanges partial or full ordering is accomplished.

Diagnostics

The macro displays error messages in the following situations:

- The input matrices have incompatible dimensions
- There are too many input arguments

- An unknown option is encountered

See also

`pencan` Kronecker canonical form of a matrix pencil

`cgivens1` Givens rotation

rank

Purpose Rank of a polynomial matrix

Syntax

```
rank(A[,tol])  
rank(A,'fft'[,tol])  
rank(A,'sylv'[,tol])
```

Description The commands

```
rank(A)  
rank(A,'fft')
```

provide an estimate of the number of linearly independent rows or columns of a polynomial matrix A by computing the rank of A evaluated at a set of Fourier points. The numbers

```
rank(A,tol)  
rank(A,'fft',tol)
```

equal the minimal number of singular values of A evaluated at the set of Fourier points that are larger than `tol`. The default value is `tol = max(size(A)) * norm(A) * eps`.

The commands

```
rank(A,'sylv')
```

```
rank(A,'sylv',tol)
```

estimate the rank of the polynomial matrix A by computing ranks of appropriate Sylvester matrices. This algorithm is based on evaluation of the rank of constant matrices and therefore the optional input argument `tol` has the same meaning as for the standard MATLAB function `rank`.

Examples

The 4×5 polynomial matrix of degree 4

```
P = prand(2,4,3)*prand(2,3,5)
```

```
Polynomial matrix in s: 4-by-5, degree: 4
```

```
P =
```

```
Column 1
```

```
-0.25 - 0.16s + 2s^2 - 0.91s^3 + 1.2s^4
```

```
-0.36 - 0.55s + 0.93s^2 - 0.88s^3 - 1.3s^4
```

```
0.11 + 0.95s - 1.1s^2 - 1.2s^3 - 1.2s^4
```

```
0.37 + 0.13s + 1.9s^2 - 1.9s^3 + 0.82s^4
```

```
Column 2
```

```
1.6 - 1.6s - 0.42s^2 - 0.31s^3 + 0.62s^4
```

```
1.6 - 0.67s - 4.8s^2 + 1.2s^3 - 0.84s^4
-0.73 - 2.6s + 0.36s^2 - 1.1s^3 + 0.87s^4
-0.73 + 1.1s + 2.2s^2 - 0.49s^3 + 0.35s^4
```

Column 3

```
-1.7 - 0.1s - 0.13s^2 + 0.096s^3 + 1.9s^4
-1.2 - 0.16s + 0.17s^2 + 1.4s^3 - 1.6s^4
1.3 + 1.3s + 5.9s^2 - 2.1s^3 + 2.4s^4
-0.49 - 0.68s + 1.2s^2 - 1.2s^3 + 0.77s^4
```

Column 4

```
0.05 + 0.25s + 0.64s^2 - 1.4s^3 + 0.41s^4
-0.22 + s + 1.2s^2 - 0.66s^3 - 1.3s^4
-0.1 - 0.37s + 1.1s^2 - 2.2s^3 - 0.27s^4
0.59 - 2.2s + 1.4s^2 - 0.097s^3 + 0.58s^4
```

Column 5

```
-2.7 + 2s + 1.3s^2 - 0.31s^3 + 0.82s^4
-2.5 - 0.082s - 2.9s^2 - 3.4s^3 - 0.0042s^4
```


$$3.4 - 2.9s + 1.7s^2 - 1.3s^3 + 0.14s^4$$

$$-0.012 + 1.3s + 3.5s^2 - 0.099s^3 + 0.18s^4$$

should have rank 3. Indeed,

```
rank(P)
```

correctly returns

```
ans =
```

```
3
```

Likewise,

```
rank(P,'sylv')
```

although it takes longer, returns

```
ans =
```

```
3
```

Algorithm

The FFT algorithm works as follows. Let P be a polynomial matrix of size $m \times n$ and degree d . First the maximal number $N = d \min(n, m)$ of finite roots is computed, and rounded up to the nearest integral power of 2. Then P is evaluated at the N Fourier points $s_i = \exp(2\pi j i / N)$, $i = 0, 1, \dots, N-1$, using the fast radix-2 FFT routine. This way the set $\{\mathbf{y}_i, \forall i = P(\mathbf{s}_i), i = 0, 1, \dots, N-1\}$ is obtained. The rank r of P may be ascertained as

$$r = \max_i \text{rank}(Y_i)$$

The procedure may be terminated if Y_i is of full rank for some i less than N , concluding that P is of full rank as well. As a result, if P has full rank then just one rank evaluation suffices as a rule, while for a singular matrix all the N rank evaluations must be performed to be sure of the rank.

The optional tolerance `tol` is used for the rank evaluations of the constant matrices Y_i using the standard MATLAB macro `rank`.

This algorithm is described in Hromčík and Sebek (1998a).

The Sylvester resultant algorithm is as follows. Let $cd(i)$ and $rd(i)$ denote the column and row degrees of the $m \times n$ matrix P , respectively. Define

$$d = \min(c, r)$$

where

$$c = cd(1) + cd(2) + \dots + cd(n) - \min_i cd(i)$$

$$r = rd(1) + rd(2) + \dots + rd(n) - \min_i rd(i)$$

Then it can be shown that

$$\text{rank } P = \min(r, c)$$

where if $d > 0$

$$rc = \text{rank sylv}(A, d) - \text{rank sylv}(A, d-1)$$

$$rr = \text{rank sylv}(A', d) - \text{rank sylv}(A', d-1)$$

and if $d = 0$

$$rc = \text{rank sylv}(A, d)$$

$$rr = \text{rank sylv}(A', d)$$

This algorithm is described in Henrion and Sebek (1998a).

Diagnostics

The macro issues an error message if an unknown command option is encountered.

See also

issingular test polynomial matrix for singularity

det determinant of a polynomial matrix

rat2lmf, rat2rmf, tf2lmf, tf2rmf

Purpose	Conversion of a rational or transfer matrix to a left or right polynomial matrix fraction
Syntax	<pre>[N,D] = rat2lmf(num,den[,tol])</pre> <pre>[N,D] = rat2rmf(num,den[,tol])</pre> <pre>[N,D] = tf2lmf(num,den[,tol])</pre> <pre>[N,D] = tf2rmf(num,den[,tol])</pre>
Description	<p>Given two polynomial matrices num and den of the same dimensions that contain the numerators and denominators of a rational transfer matrix the function</p> <pre>[N,D] = rat2lmf(num,den)</pre> <p>converts the transfer matrix to the left coprime polynomial matrix fraction $D^{-1}N$. The polynomial matrices N and D inherit the indeterminate variable of num and den.</p> <p>Given two cell arrays num and den of the same dimensions that contain the numerators and denominators of a rational transfer matrix in MATLAB format the function</p> <pre>[N,D] = tf2lmf(num,den)</pre>

converts the transfer matrix to the left coprime polynomial matrix fraction $D^{-1}N$.
The polynomial matrices N and D inherit the default indeterminate variable.

Similarly, the functions

```
[N,D] = rat2rmf(num,den)
```

```
[N,D] = tf2rmf(num,den)
```

converts the transfer matrix to the right coprime polynomial matrix fraction ND^{-1} .

In all cases a tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Transfer function models are not recommended. State space and matrix fraction models are more natural and the conversion from and to transfer function models is numerically delicate.

Examples

Consider the transfer matrix

$$\begin{bmatrix} \frac{-5}{s-1} \\ \frac{s^2 - 5s + 6}{s^2 + s} \end{bmatrix}$$

First define the matrices that contain the numerator and denominator polynomials in Polynomial Toolbox format:

```
num = [      -5
```

```

        s^2-5*s+6 ];
den = [      s-1
        s^2+s   ];

```

A left matrix fraction representation of the transfer matrix follows by typing

```
[Nl,Dl] = rat2lmf(num,den)
```

We obtain

```

Nl =
    -5
    6 - 5s + s^2

Dl =
    -1 + s      0
    0          s + s^2

```

A right fraction follows similarly by the command

```
[Nr,Dr] = rat2rmf(num,den)
```

```

Nr =
    -5s - 5s^2

```

$$-6 + 11s - 6s^2 + s^3$$

$D_r =$

$$-s + s^3$$

Algorithm

To obtain a left matrix fraction the routines `lrm` and `rdiv` are employed to bring all entries of the transfer matrix H on a least common denominator d . As a result H is of the form

$$H = N_o / d$$

with N_o a polynomial matrix. After this the right fraction

$$N_o(dI)^{-1}$$

is converted to a coprime left fraction using `rmf2lmf`.

The macros `rat2rmf` and `tf2rmf` are the duals of `rat2lmf` and `tf2lmf`.

Diagnostics

The macros return error messages if

- The tolerance is invalid
- The number of input arguments is incorrect
- The input matrices have incompatible sizes
- Illegal input for the denominator is encountered

See also

`lmf2rat`, `rmf2rat`

`lmf2tf`, `rmf2tf`

`tf`

conversion to a left or right polynomial matrix fraction to a rational or transfer matrix

create a Control System Toolbox LTI object in transfer function format

real

Purpose	Real part of a polynomial matrix
Syntax	<code>x = real(z)</code>
Description	For a polynomial matrix Z , the command <code>x = real(z)</code> returns the polynomial matrix X whose coefficients are the real parts of the corresponding coefficients of Z .
Examples	<p>If</p> <pre>P = [(1+i)*s (1-i)*s+s^2];</pre> <p>then</p> <pre>real(P)</pre> <p>results in</p> <pre>ans =</pre> $\begin{bmatrix} s & s + s^2 \end{bmatrix}$
Algorithm	The macro <code>real</code> uses standard MATLAB operations.
Diagnostics	The macro displays no error messages.

See also

<code>imag</code>	imaginary part of a polynomial matrix
<code>ctranspose (')</code>	complex conjugate transpose of a polynomial matrix
<code>conj</code>	conjugate of a polynomial matrix

reverse

Purpose Reverse the variable of a polynomial matrix fraction

Syntax

```
[N,D] = reverse(P,Q[,tol])
[N,D] = reverse(P,Q,'l'[,tol])
[N,D] = reverse(P,Q,'r'[,tol])
```

Description Given a square nonsingular polynomial matrix Q and a polynomial matrix P with the same number of rows the commands

```
[N,D] = reverse(P,Q)
[N,D] = reverse(P,Q,'l')
```

compute matrices N and D such that

$$D^{-1}(s)N(s) = Q^{-1}(1/s)P(1/s)$$

If P and Q are left coprime then so are N and D . The command

```
[N,D] = reverse(P,Q,'r')
```

computes the matrices N and D such that

$$N(s)D^{-1}(s) = P(1/s)Q^{-1}(1/s)$$

The commands also work for other variables than s . N and D inherit the variables of P and Q .

The optional input argument `tol` specifies a tolerance. Its default value is the global zeroing tolerance.

Examples

The command may be used to convert transfer matrices in the forward shift operator z to transfer matrices in the backward shift operator z^{-1} , and vice-versa. Consider by way of example the polynomials

$$P = 2 + 3z; \quad Q = 4 + z^2;$$

that define the transfer function

$$H(z) = \frac{P(z)}{Q(z)} = \frac{2 + 3z}{4 + z^2}$$

Then

$$\begin{aligned} [N, D] &= \text{reverse}(P, Q); \\ N.\text{var} &= 'z^{-1}', \quad D.\text{var} = 'z^{-1}' \end{aligned}$$

produces

$$\begin{aligned} N &= \\ &3z^{-1} + 2z^{-2} \\ D &= \end{aligned}$$

$$1 + 4z^{-2}$$

These polynomials define the transfer function

$$H(z^{-1}) = \frac{N(z^{-1})}{D(z^{-1})} = \frac{3z^{-1} + 2z^{-2}}{1 + 4z^{-2}}$$

As another application, suppose that we wish to determine the number of poles and zeros that the unimodular polynomial matrix

$$U(s) = \begin{bmatrix} s & 1 \\ -1+s & 1 \end{bmatrix}$$

has at infinity. To this end we determine a coprime factorization of $U(1/s)$ and determine how many poles and zeros this fraction has at 0. Writing

$$U(s) = Q^{-1}(s)P(s), \quad Q(s) = I, \quad P(s) = U(s)$$

we may compute the desired fraction

$$U(1/s) = D^{-1}(s)N(s)$$

according to

```
U = [s 1; -1+s 1];
[N,D] = reverse(eye(2),U)
N =
```

$$D = \begin{bmatrix} 0 & s \\ 1 & -1 \\ 1 - s & s \\ 1 & 0 \end{bmatrix}$$

Inspection shows that the numerator polynomial N has one root at 0, so that U has exactly one zero at infinity. Also the denominator D has a root at 0. Hence, U has exactly one pole at infinity.

Algorithm

Given the left fraction $Q^{-1}(s)P(s)$ the polynomial matrix $[Q \ P]$ is first unimodularly transformed from the left to make it row reduced. After replacing s with $1/s$ in the row reduced matrix that is obtained this way each row is multiplied by the smallest power of s needed to make the row polynomial. The result is partitioned as $[D \ N]$.

Diagnostics

The routine displays error messages under the following circumstances:

- There are not enough input arguments
- The denominator matrix is not square
- An unknown command option is encountered
- The numerator and denominator sizes do not match

A warning follows if

- The denominator matrix is singular to working precision
- The input matrices have inconsistent variables

See also

`lmf2ss, rmf2ss` conversion of left or right matrix fraction representation to (generalized) state space representation

`lmf2rmf, rmf2lmf` conversion of left matrix fraction representation to right matrix fraction and vice-versa

roots

Purpose

Roots of a polynomial matrix

Syntax

```
rts = roots(P[,tol])
```

```
rts = roots(P,'det'[,tol])
```

```
rts = roots(P,'eig'[,tol])
```

```
rts = roots(P,'all'[,tol])
```

```
rts = roots(P,'det','all'[,tol])
```

```
rts = roots(P,'eig','all'[,tol])
```

Description

The command

```
rts = roots(P)
```

computes the finite roots of a polynomial matrix P , that is, those complex values of s for which $P(s)$ loses rank.

If P is square nonsingular then the commands

```
rts = roots(P,'det')
```

```
rts = roots(P,'eig')
```


allow the user to specify the algorithm that is used. With the option `'det'` the roots are computed as the roots of determinant of P . This is the default method. With the option `'eig'` the roots are computed as the generalized eigenvalues of the block companion matrix pair associated with P .

With the option `'all'` the infinite roots of P are returned as well. If the option `'eig'` is used then all computed eigenvalues are returned without filtering out the eigenvalues computed as `Inf`, `NaN` or a large number. Otherwise the number of infinite eigenvalues is computed as the number of roots at 0 of $P(1/s)$.

The optional input parameter `tol` is used for rank testing. Also, any root whose magnitude is greater than $1/\text{tol}$ is taken to be infinite. The default value for `tol` is the global zeroing tolerance.

Examples

The command

```
P = prand(3,2,2,'sta','int')
```

generates the stable 2×2 polynomial matrix P with 3 roots

```
P =
      -5 - 6s - s^2      -22 + 6s + s^2
      -1.4e+002 - 27s      1.4e+002 + 28s
```

Indeed,

```
roots(P)
```

returns

```
ans =  
-1.0000 +27.0000i  
-1.0000 -27.0000i  
-5.0000
```

The command

```
roots(P,'eig','all')
```

returns

```
ans =  
NaN - NaNi  
-1.0000 -27.0000i  
-1.0000 +27.0000i  
-5.0000 + 0.0000i
```

while by typing

```
roots(P,'all')
```

we find that the matrix actually has no roots at infinity because it is row reduced:

```
ans =
-1.0000 +27.0000i
-1.0000 -27.0000i
-5.0000
```

The polynomial matrix

$$P(s) = \begin{bmatrix} s & 1 & 1 & 1 & 1 \\ 1 & s & 1 & 1 & 1 \\ 1 & s & 1 & 1 & 1 \end{bmatrix}$$

obviously has normal rank 2 but rank 1 at $s = 1$. Indeed, typing

```
P = [ s  1  1  1  1
      1  s  1  1  1
      1  s  1  1  1 ];
roots(P)
```

confirms that the matrix has a root at $s = 1$:

```
ans =
1.0000
```

The command

```
U = prand(2,2,2,'uni')
```

generates the unimodular matrix

```
U =
      2.1 + 0.37s - 0.23s^2      1.3 - 0.4s
      0.86 + 0.57s              1
```

The command

```
roots(U,'all')
```

reveals that U has two roots at infinity:

```
ans =
      Inf
      Inf
```

but of course no finite roots.

Algorithm

If the `'det'` option is used then the roots are computed as the roots of $\det(P)$.

If the `'eig'` option is used then the roots are computed as the generalized eigenvalues of the companion matrix pair associated with P (see `companion`).

If the matrix is rectangular or singular then the algorithm proceeds as follows. If P has size $n \times m$ and rank r then its roots are also roots of

$$d(s) = \det(\text{rand}(r,n)P(s)\text{rand}(m,r))$$

where $\text{rand}(i, j)$ is an $i \times j$ random constant matrix. By checking which of the roots of d are also roots of another instance of d the roots of P are identified.

If the option '**all**' is used then except if the option '**eig**' is also used first a coprime factorization $D^{-1}(s)N(s) = P(1/s)$ is found with the help of the routine **reverse**. The number of zeros of N is then computed by checking how many zeros $\det(\text{rand}(r,n)N(s)\text{rand}(m,r))$ has at 0.

Diagnostics

The macro issues an error message if an unknown command option is encountered.

See also

det determinant of a polynomial matrix
rank rank of a polynomial matrix

rot90

Purpose Rotate a polynomial matrix by 90°

Syntax `B = rot90(A)`

`B = rot90(A,k)`

Description The command

`B = rot90(A)`

rotates the matrix A counter-clockwise by 90 degrees. The command

`B = rot90(A,k)`

rotates the matrix A counter-clockwise by $k \times 90$ degrees, where k is an integer.

Examples The commands

```
P = [ 1+s      2*s^2
      -2      3+4*s^3
      5      6+7*s  ];
```

```
Q = rot90(P)
```

produce the response

$$Q = \begin{bmatrix} 2s^2 & 3 + 4s^3 & 6 + 7s \\ 1 + s & -2 & 5 \end{bmatrix}$$

while

```
Q = rot90(P,2)
```

results in

$$Q = \begin{bmatrix} 6 + 7s & 5 \\ 3 + 4s^3 & -2 \\ 2s^2 & 1 + s \end{bmatrix}$$

Algorithm

The routine uses standard routines from the Polynomial Toolbox.

Diagnostics

The macro displays error messages if it has too many input or output arguments, or if k is not a scalar.

See also

`fliplr` flip a polynomial matrix left-right
`flipud` flip a polynomial matrix up-down

schurst

Purpose Ordered Schur decomposition

Syntax `[S,U,ind] = schurst(C)`
`[S,U,ind] = schurst(C,tol)`

Description The command

`[S,U,ind] = schurst(C)`

computes the ordered complex Schur decomposition $C = USU'$ of the constant matrix C .

The constant matrix U is unitary ($UU' = I$) and S is upper triangular. The eigenvalues of C with negative real part appear first along the diagonal of S .

The vector `ind` contains the corresponding column indices of U so that the columns of $U(:,ind)$ span the maximal stable invariant subspace of C .

An optional tolerance argument `tol` may be specified. It is used to determine the sign of the diagonal elements.

Examples Consider the constant matrix

```
C = [ 4  -3  11
      -1  1   5
```



```
0 0 -4 ];
```

The command line

```
[S,U,ind] = schurst(C)
```

results in

S =

```
-4.0000    6.2685   10.4206
         0    4.7913    1.4547
    0.0000   -0.0000    0.2087
```

U =

```
0.7433    0.6576    0.1231
0.5415   -0.6994    0.4665
-0.3929    0.2801    0.8759
```

ind =

```
1
0
0
```

Hence, the maximal stable invariant subspace of C is spanned by the columns of

```
U(:,ind)

ans =

    0.7433
    0.5415
   -0.3929
```

Algorithm

First the complex Schur decomposition of C is computed with the MATLAB standard functions `schur` and `rsf2csf`. After this the diagonal elements of S are sorted by application of successive Givens rotations on S and U (Golub and Van Loan, 1996). If the tolerance `tol` is not specified its value is taken as $10 \cdot \text{eps} \cdot \max(\text{abs}(\text{diag}(T)))$, where T is the (not necessarily ordered) Schur form of C as returned by `schur`.

Diagnostics

The macro returns an error message if the input matrix is not square.

See also

`qzord` ordered QZ decomposition

spf, spcof

Purpose Polynomial spectral factorization and co-factorization

Syntax

```
[A,J] = spf(B[,tol])
```

```
[A,J] = spf(B,'ext'[,tol])
```

```
[A,J] = spf(B,'are'[,tol])
```

```
[A,J] = spf(B,'ext','nnc'[,tol])
```

```
[A,J] = spf(B,'are','nnc'[,tol])
```

```
A = spf(B,'sylv'[,tol])
```

```
A = spf(B,'red'[,tol])
```

```
[A,J] = spcof(B[,tol])
```

All options of **spcof** follow those of **spf**

Description

If B is a continuous-time para-Hermitian polynomial matrix, that is, B is a square polynomial matrix in the variable s (or p) such that

$$B(s) = B^T(-s)$$

then the command

[A,J] = spf(B)

determines a stable square polynomial matrix A (stable in the continuous-time sense — see **isstable**) and a signature matrix J such that

$$B(s) = A^T(-s)JA(s)$$

The signature matrix J is diagonal of the form

$$J = \text{diag}(1, 1, \dots, 1, -1, -1, \dots, -1)$$

If B is diagonally reduced then the column degrees of the spectral factor A equal the half diagonal degrees of B .

A tolerance **tol** may be specified as an additional input argument. Its default value is the global zeroing tolerance.

The command

[A,J] = spf(B,'ext')

performs factorization with factor extraction by interpolation. This is the default method. A tolerance of **tol***norm(C ,1) is used for ordering the complex Schur decomposition of the companion matrix C corresponding to B .

The command

[A,J] = spf(B,'are')

uses an algorithm based on a state space solution that involves solving an algebraic Riccati equation. The tolerance `tol` is used in the macro `axb` for recovering the spectral factor.

The commands

```
[A,J] = spf(B,'ext','nnc')
```

```
[A,J] = spf(B,'are','nnc')
```

perform a “nearly non-canonical factorization,” that is, the factorization takes the form

$$B(s) = A^T(-s)J^{-1}A(s)$$

J is diagonal with the diagonal entries arranged according to decreasing value. If B does not have a canonical factorization then both the spectral factor A and the matrix J are singular, and the spectral factorization should be interpreted as the identity

$$A(s)B^{-1}(s)A^T(-s) = J$$

If B is positive-definite on the imaginary axis then the commands

```
A = spf(B,'syl')
```

```
A = spf(B,'red')
```

implement a Newton-Raphson iterative scheme involving the corresponding implementation of the function **axxab**. The signature matrix in this case is the identity matrix. The iterative process is terminated when $\text{norm}(A' * A - B)$ is less than $\text{tol} * \text{norm}(B) * 100$. The tolerance **tol** is also used in the macro **axxab**.

In the discrete-time case, that is, if the matrix B is a square polynomial matrix in the variable z (or q , d or z^{-1}), then the matrix B is para-Hermitian if

$$B(z) = B^T(1/z)$$

In this case B is not a polynomial matrix but is of the form

$$B(z) = B_d^T z^{-d} + B_{d-1}^T z^{-d+1} + \dots + B_1^T z^{-1} + B_0 + B_1 z + \dots + B_d z^d$$

B needs to be provided to the macro **spf** in the polynomial form

$$B(z) := z^d B(z)$$

The degree offset d is evaluated upon cancelation of the leading and trailing zero matrix coefficients.

In the discrete-time only factorizations are performed where B is positive-definite on the unit circle so that the signature matrix J is the unit matrix. In this case the command

$$A = \text{spf}(B)$$

determines a polynomial matrix A such that A is stable (in the discrete-time sense — see `isstable`) and

$$B(z) = A^T(1/z)A(z)$$

The algorithm uses a Newton-Raphson iterative scheme. The solution has its constant coefficient matrix $A(0)$ upper triangular with positive diagonal entries. In the form

```
A = spf(B,'syl')
```

the macro is based on a Sylvester matrix algorithm that in turn relies on the macro `axxab`. This is the default method. In the form

```
A = spf(B,'red')
```

the solution is based on polynomial reductions. The iterative scheme is stopped when $\text{norm}(A^*A - B)$ is less than $\text{tol} * \text{norm}(B) * 100$. The tolerance `tol` is also used in the macro `axxab`.

If B is a constant symmetric matrix then A is also constant and is obtained together with J from the Schur decomposition of B .

The macro `spcof` implements spectral co-factorization, that is, given a continuous-time para-Hermitian matrix B the command

```
[A,J] = spcof(B)
```

computes a stable square polynomial matrix A (the spectral co-factor) and a signature matrix J such that

$$B(s) = A(s)JA^T(-s)$$

The options of `spcof` follow those of `spf` with obvious modifications where needed.

Examples

Let

$$B = \begin{bmatrix} 1-4s^2 & 2-4s & 0 \\ 2+4s & 8-4s^2 & -1-2s \\ 0 & -1+2s & 1-4s^2 \end{bmatrix};$$

Its spectral factorization follows from

$$[A, J] = \text{spf}(B)$$

as

$$A = \begin{bmatrix} 1 + 2s & 2 & 0 \\ 0 & 1.7 + 2s & 0 \\ 0 & -1 & 1 + 2s \end{bmatrix}$$

$$J =$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Next consider

$$B = \begin{bmatrix} -s^2 & -1-s \\ -1+s & -3 \end{bmatrix};$$

Its spectral factorization follows as

$$[A, J] = \text{spf}(B)$$

$$A =$$

$$\begin{bmatrix} -0.67 - 1.1s & -0.26 \\ -0.67 + 0.41s & -1.8 \end{bmatrix}$$

$$J =$$

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The next example

$$B = \begin{bmatrix} 1 & 0.5+s^2 \end{bmatrix}$$

```
0.5+s^2  1+s^4  ];
```

involves a matrix that is not diagonally reduced. The diagonal leading coefficient matrix

```
lcoef(B,'dia')
```

```
ans =
```

```
1      1
1      1
```

is obviously singular. We obtain the spectral factorization

```
[A,J] = spf(B)
```

```
A =
```

```
1      0.5 + s^2
0      0.87 + s
```

```
J =
```

```
1      0
0      1
```

Next, consider the para-Hermitian matrix

$$B(s) = \begin{bmatrix} e & -0.5 + 0.5s \\ -0.5 - 0.5s & 0.75 + s^2 \end{bmatrix}$$

For $e = 0$ the spectral factorization of B is noncanonical. We study the spectral factorization of

$$B = \begin{bmatrix} 1e-6 & -0.5+0.5*s \\ -0.5-0.5*s & 0.75+s^2 \end{bmatrix};$$

The command

```
[A,J] = spf(B)
```

results in

```
Warning: SPF: Factorization close to noncanonical.
```

```
Consider nearly non-canonical factorization.
```

```
A =
```

$$\begin{bmatrix} -0.42 & 4.2e+005 - 0.17s \\ 0.42 & -4.2e+005 - s \end{bmatrix}$$

```
J =
```

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Following the advice we now try

```
[A,J] = spf(B,'nnc')
```

This yields

```
A =
    1.4e-006    -0.71 + 0.71s
         0      0.71 + 0.71s

J =
    1.0e-005 *
    0.2000         0
         0    -0.2000
```

Note that the large numbers have disappeared but that both A and J are near-singular. Also note that the co-factorization of this same matrix B

```
[C,J] = spcof(B)
```

is perfectly well-behaved:

```
C =
   -0.42         -0.42
```

$$J = \begin{bmatrix} 0.91 + 0.17s & -0.28 - s \\ 1 & 0 \\ 0 & -1 \end{bmatrix}$$

We conclude with a discrete-time example where

$$B(z) = \begin{bmatrix} z + 6 + z^{-1} & 1 \\ 1 & 2 \end{bmatrix}$$

We need to present the matrix to `spf` as

$$B = \begin{bmatrix} z^2 + 6z + 1 & z \\ z & 2z \end{bmatrix};$$

The spectral factorization command

$$A = \text{spf}(B)$$

returns

$$A = \begin{bmatrix} 0.41 + 2.4z & 0.41z \\ -0.13 & 1.4z \end{bmatrix}$$

Algorithm

If in the continuous-time case the matrix is not diagonally reduced then it is first made diagonally reduced using **diagred**.

The “interpolation” algorithm (option **'int'**, the default method) relies on spectral factor extraction as described in Algorithm 7.1 of Kwakernaak and Sebek (1994). The “Riccati” algorithm (option **'are'**) is based on Algorithm 8.1–2. The **'nnc'** variant is described in Section IX of this same paper.

The Raphson-Newton scheme for positive-definite factorizations follows Jezek and Kucera (1985), where also the polynomial reduction algorithm (option **'red'**) is described.

Diagnostics

The macro issues an error message under the following conditions:

- Invalid Not-a-Number or Inf entry are found in the input matrix
- The input matrix is not square, not para-Hermitian or singular
- An invalid option or input argument is encountered
- The input matrix is not positive-definite on the imaginary axis or on the unit circle as needed
- The factorization fails in any of several ways

See also

plqg polynomial solution of the LQG problem
mixedss solution of the mixed sensitivity problem

splqg

Purpose

Polynomial solution of SISO LQG problems

Syntax

`[y,x,regpoles,obspoles] = splqg(d,n,p,q,rho,mu)`

Description

The function call

`[y,x,regpoles,obspoles] = splqg(d,n,p,q,rho,mu)`

results in the solution of the SISO LQG problem defined by

- Response of the measured output to the control input:

$$y = P(s)u, \quad P(s) = \frac{n(s)}{d(s)}$$

- Response of the controlled output to the control input:

$$z = Q(s)u, \quad Q(s) = \frac{p(s)}{d(s)}$$

- Response of the measured output to the disturbance input:

$$y = R(s)v, \quad R(s) = \frac{q(s)}{d(s)}$$

In state space form

$$\begin{aligned}\dot{x} &= Ax + Bu + Gv & P(s) &= C(sI - A)^{-1}B \\ z &= Dx & Q(s) &= D(sI - A)^{-1}B \\ y &= Cx + w & R(s) &= C(sI - A)^{-1}G\end{aligned}$$

The scalar white state noise v has intensity 1, and the white measurement noise w has intensity μ . The compensator $C(s) = y(s)/x(s)$ minimizes the steady-state value of

$$E\{z^2(t) + ru^2(t)\}$$

The output argument **regpoles** contains the regulator poles and **obspoies** contains the observer poles. Together the regulator and observer poles are the closed-loop poles.

Examples

Consider the LQG problem for the plant with transfer function

$$P(s) = \frac{10^{-4}(s^4 + 0.16s^3 + 10.0088s^2 + 0.4802s + 9.0072)}{s^2(s^4 + 0.08s^3 + 2.5022s^2 + 0.06002s + 0.56295)}$$

This is a scaled version of a mechanical positioning system discussed by Dorf (1989, pp. 544–546). The definition of the LQG problem is completed by choosing $Q = R = P$ so that $p = q = n$. Furthermore, we let $r = 1$ and $m = 10^{-6}$.

```
n = 1e-4*(s^4+0.16*s^3+10.0088*s^2+0.4802*s+9.0072);
d = s^2*(s^4+0.08*s^3+2.5022*s^2+0.06002*s+0.56295);
```



```
p = n; q = n; rho = 1; mu = 1e-6;
```

We can now compute the optimal compensator:

```
[y,x,regpoles,obspoles] = splqg(d,n,p,q,rho,mu)
```

MATLAB returns

```
y =
```

```
0.9 + 34s + 7.5s^2 + 1.5e+002s^3 + 6.4s^4 + 61s^5
```

```
x =
```

```
1 + 2.7s + 4.8s^2 + 5.5s^3 + 4.4s^4 + 1.9s^5 + s^6
```

```
regpoles =
```

```
-0.0300 + 1.5000i
```

```
-0.0300 - 1.5000i
```

```
-0.0101 + 0.5000i
```

```
-0.0101 - 0.5000i
```

```
-0.0284 + 0.0282i
```

```
-0.0284 - 0.0282i
```

```
obspoles =
```

$$-0.0692 + 1.5048i$$

$$-0.0692 - 1.5048i$$

$$-0.6931 + 0.3992i$$

$$-0.6931 - 0.3992i$$

$$-0.1734 + 0.7684i$$

$$-0.1734 - 0.7684i$$

Algorithm

The regulator characteristic polynomial f_r and the observer characteristic polynomial f_o are found by two polynomial spectral factorizations from

$$f_r^* f_r = d^* d + \frac{1}{r} p^* p$$

$$f_o^* f_o = d^* d + \frac{1}{m} q^* q$$

The closed-loop characteristic polynomial may now be obtained as $f = f_r f_o$. The compensator $C = y/x$ follows by solving

$$f = dx + ny$$

for x and y such that y/x is proper.

Diagnostics

The macro displays error messages in the following situations:

- n/d is not proper

- p/d is not proper
- q/d is not proper

See also

`plqg` solution of MIMO LQG problems

`mixedds` solution of the mixed sensitivity problem for SISO systems

ss

Purpose Create state space models or convert LTI model to state space form

Syntax

```
sys = ss(N,D)

sys = ss(N,D,'l')

sys = ss(N,D,'r')

sys = ss(N,D,T[,tol])

sys = ss(N,D,'l',T[,tol])

sys = ss(N,D,'r',T[,tol])
```

Description

The command **ss** is a function from the MATLAB Control System Toolbox that creates state space system objects, or converts other LTI objects to state space form. A system is in state space form if it is described by equations of the type

$$\dot{x} = ax + bu$$

$$y = cx + du$$

with a , b , c , and d constant matrices. Polynomial matrices d are not supported in Control System Toolbox state space objects.

The function `ss` has been overloaded so that in addition to its Control System Toolbox functionality it converts systems represented in left or right polynomial matrix fraction form to a Control System Toolbox LTI object in state space form.

Let D be a square nonsingular row reduced polynomial matrix and N a polynomial matrix with the same number of rows as D such that

$$H = D^{-1}N$$

is proper. Then the commands

```
sys = ss(N,D)
```

```
sys = ss(N,D,'l')
```

convert the system with transfer matrix H to a Control System Toolbox LTI object in state space form.

If N and D are polynomial matrices in s or p then a continuous-time LTI object is created. If they are matrices in z , q , z^{-1} or d then a discrete-time object with sampling time $T = 1$ is created. In the latter case the commands

```
sys = ss(N,D,T)
```

```
sys = ss(N,D,'l',T)
```

with T a real number, create a discrete-time system with sampling time T . If $T = 0$ then the object is taken to be a continuous-time system.

If the option `'1'` is replaced with `'x'` then N and D define the system whose transfer matrix is the right fraction

$$H = ND^{-1}$$

D needs to be square column reduced with the same number of columns as N , and H needs to be proper.

The optional input parameter `tol` defines a tolerance that is used in the conversion of the matrix fraction to state space form. Note that this parameter may only be included if also the input parameter T is included.

The function `ss` only works if the Control System Toolbox is included in the MATLAB path.

Later versions of the Polynomial Toolbox will support a polynomial matrix fraction LTI object.

Examples

If

```
D = [ 2    1+s
      s^2   3 ];

N = [ 1
      1 ];
```

then typing

```
sys = ss(N,D)
```

results in

```
a =
```

	x1	x2	x3
x1	-1	-2	0
x2	0	0	1
x3	-3	0	0

```
b =
```

	u1
x1	1
x2	0
x3	1

```
c =
```

	x1	x2	x3
y1	0	1	0
y2	1	0	0


```
x1
      y1      1
d =
      u1
      y1      0
Sampling time: 1
Discrete-time system.
```

Algorithm

The function calls `lmf2ss` or `rmf2ss`.

Diagnostics

The function returns an error message in the following circumstances:

- The Control System Toolbox is not included in the MATLAB path
- N and D are polynomial matrices in different variables
- The third or fourth input argument are incorrect
- The matrix fraction defined by N and D is not proper.

See also

lti2lmf, lti2rmf	conversion of a state space LTI object to a left or right matrix fraction model
-------------------------	---

tf	creation of a transfer function LTI object or conversion to a transfer function model
-----------	---

zpk	creation of a zero-pole-gain LTI object or conversion to a zero-pole-gain model
lmf2ss, rmf2ss ss2lmf, ss2rmf	conversion from a left or right polynomial matrix fraction representation to a state space model, and vice-versa

ss2dss

Purpose Conversion of state space to descriptor representation

Syntax `[a,b,c,d,e] = ss2dss(A,B,C,D)`

`[a,b,c,d,e] = ss2dss(A,B,C,D,tol)`

Description The commands

`[a,b,c,d,e] = ss2dss(A,B,C,D)`

`[a,b,c,d,e] = ss2dss(A,B,C,D,tol)`

with D a constant matrix or a polynomial matrix in s , convert the state space system

$$\dot{x} = Ax + Bu$$

$$y = Cx + D(s)u$$

with s the derivative operator given by $su(t) = du(t)/dt$, to the descriptor system

$$e\dot{x} = ax + bu$$

$$y = cx + du$$

The systems are equivalent in the sense that

$$C(sI - A)^{-1}B + D(s) = c(se - a)^{-1}b + d$$

The d matrix of the descriptor system equals the constant term of the polynomial matrix D .

If D is a polynomial matrix in p then the system is also taken to be a continuous-time system. If D is a polynomial in z or q then the input defines the discrete-time system

$$x(t+1) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + D(z)u(t)$$

with z the time-shift operator given by $zu(t) = u(t+1)$. In this case the output parameters define the discrete-time descriptor system

$$ex(t+1) = ax(t) + bu(t)$$

$$y(t) = cx(t) + du(t)$$

If D is polynomial in z^{-1} or d then an error message follows.

The optional parameter `tol` is a tolerance that is used in the row reduction needed to find a minimal descriptor representation for the polynomial part of the system. Its default value is `eps*10^2`.

Example

We consider the degenerate state space system defined by

```
A = []; B = ones(0,1); C = ones(1,0); D = s^2;
```

Inspection shows that this system has the purely polynomial transfer function

$$H(s) = s^2$$

Application of

```
[a,b,c,d,e] = ss2dss(A,B,C,D)
```

yields

```
a =
```

```
    1    0    0
    0    1    0
    0    0    1
```

```
b =
```

```
    0
    0
    1
```

```
c =
```

```
   -1    0    0
```

```
d =
```

$$e = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Algorithm

Denoting

$$D(s) = D_0 + \underbrace{D_1s + D_2s^2 + \dots + D_p s^p}_{\hat{D}(s)}$$

the state space system may be represented as the parallel connection of a system with the proper state space representation

$$\dot{x} = Ax + Bu$$

$$y = Cx + D_0u$$

and a system whose transfer function is the polynomial matrix $\hat{D}(s)$. A descriptor representation of this second system is obtained by determining a minimal state representation of the system with the (strictly proper) transfer matrix $\hat{D}(1/s) / s$. If

$$\frac{\hat{D}(1/s)}{s} = c(sI - a)^{-1}b$$

then it is easy to see that $\hat{D}(s)$ has the minimal descriptor representation

$$\hat{D}(s) = c(I - sa)^{-1}b$$

To find a minimal realization of $\hat{D}(1/s)/s$ we define $P(s) = s\hat{D}(s)$ and determine a unimodular transformation U such that

$$U[I \ P(s)] = [d(s) \ n(s)]$$

is row reduced. Note that if P is row reduced then $U = I$. It easily follows that

$$P(s) = d^{-1}(s)n(s)$$

We next multiply each row of

$$\begin{bmatrix} d(\frac{1}{s}) & n(\frac{1}{s}) \end{bmatrix}$$

by the smallest power of s needed to make the row polynomial. Denoting the result as

$$\begin{bmatrix} \hat{d}(s) & \hat{n}(s) \end{bmatrix}$$

we have

$$\frac{1}{s}\hat{D}(\frac{1}{s}) = P(\frac{1}{s}) = \hat{d}^{-1}(s)\hat{n}(s)$$

with \hat{d} and \hat{n} left coprime. After a further unimodular transformation to make \hat{d} row reduced we may use **lmf2ss** to construct a minimal state space representation.

Diagnostics

The macro issues an error message under the following conditions.

- The value of the tolerance is invalid
- The number of input arguments is incorrect
- The input matrices A , B and C are not constant matrices
- The fourth argument has an invalid format or variable symbol
- The input matrices have incompatible sizes

See also

dss2ss	conversion of descriptor representation to state representation
dss2lmf , dss2rmf lmf2dss , rmf2dss	conversion from descriptor form to left or right polynomial matrix fraction form and vice-versa
dss , ss	create descriptor state space and state space models or convert LTI model to descriptor state space or state space form

ss2lmf, ss2rmf

Purpose Conversion of state space representation to left or right matrix fraction representation

Syntax

```
[N,D] = ss2lmf(a,b,c)
[N,D] = ss2lmf(a,b,c,d)
[N,D] = ss2lmf(a,b,c,d,tol)
[N,D] = ss2rmf(a,b,c)
[N,D] = ss2rmf(a,b,c,d)
[N,D] = ss2rmf(a,b,c,d,tol)
```

Description The input arguments of the commands

```
[N,D] = ss2lmf(a,b,c)
[N,D] = ss2lmf(a,b,c,d)
[N,D] = ss2lmf(a,b,c,d,tol)
```

with d a constant matrix or a polynomial in s , define the (generalized) state space system

$$\begin{aligned}\dot{x} &= ax + bu \\ y &= cx + d(s)u\end{aligned}$$

The output arguments represent the transfer matrix

$$H(s) = c(sI - a)^{-1}b + d(s)$$

of the system as the left coprime polynomial matrix fraction

$$H(s) = D^{-1}(s)N(s)$$

so that the denominator matrix D is row reduced .

The commands

```
[N,D] = ss2rmf(a,b,c)
```

```
[N,D] = ss2rmf(a,b,c,d)
```

```
[N,D] = ss2rmf(a,b,c,d,tol)
```

produce the right polynomial matrix fraction

$$H(s) = N(s)D^{-1}(s)$$

so that D is column reduced.

If the input parameter d is omitted then it is set equal to the zero matrix.

The optional input parameter `tol` is a tolerance that is used in determining the observability or controllability indexes, that is, the row or column degrees of D . Its default value is `eps*1e2`.

The continuous-time interpretation of the input arguments applies if the input argument d is a polynomial matrix in s or p . The output arguments N and D are returned in the same variable.

If the input parameter d is a polynomial matrix in z or q then the input parameters are taken to define the discrete-time system

$$\begin{aligned}x(t+1) &= ax(t) + bu(t) \\ y(t) &= cx(t) + d(z)u(t)\end{aligned}$$

with z the time-shift operator given by $zu(t) = u(t+1)$. The output parameters then are polynomial matrices in z or q representing the discrete-time system with transfer matrix $H = D^{-1}N$ or $H = ND^{-1}$.

If d is a polynomial matrix in z^{-1} or d then an error message occurs.

If d is a constant matrix then the input matrices are taken to define a continuous-time system if the default indeterminate variable is s or p , and a discrete-time system if the default variable is z , q , z^{-1} or d . The output matrices N and D are returned as polynomial matrices in the default variable.

Examples

We consider the state space system defined by

$$a = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$

```

      0  0  0  0
      0  0 -2  0
      0  0  0  0 ];
b = [ 0  0
      1  0
      0  1
      0  0 ];
c = [ 1  0  1  0
      0  0  1  1 ];
d = [ 0  0
      0  0 ];

```

The corresponding right matrix fraction follows as

```
[N,D] = ss2rmf(a,b,c,d)
```

Constant polynomial matrix: 2-by-2

N =

```

      1      -1

```

```

0      -1
D =
s^2      0
0      -2 - s

```

Modification of d in

```

d = [ 1+s 0
      0 0 ];

```

results in

```

[N,D] = ss2rmf(a,b,c,d)

N =
1 + s^2 + s^3      -1
0                      -1

D =
s^2      0
0      -2 - s

```

If we let

```
d = [ 1+z  0
      0   0 ];
```

then

```
[N,D] = ss2lmf(a,b,c,d)
```

produces the left matrix fraction defined by

```
N =
      1 + z^2 + z^3      0
      0                  -1

D =
      z^2      -z^2
      0        -2 - z
```

Next consider the state system defined by

```
a = 1; b = 1; c = 1;
```

Then

```
gprop s, [N,D] = ss2lmf(a,b,c)
```

yields

Constant polynomial matrix: 1-by-1

$N =$

1

$D =$

$-1 + s$

while

`gprop z^-1, [N,D] = ss2lmf(a,b,c)`

results in

$N =$

z^{-1}

$D =$

$1 - z^{-1}$

Algorithm

The algorithm for `ss2lmf` is described in Srijbos (1995, 1996). Using a sequence of Householder transformations the matrix pair (a, b) is brought into block Hessenberg form and the noncontrollable part is removed (see `bhf`). Next the macro `bhf` is used on the new matrix pair (a', c') to remove the nonobservable part. From the result the left coprime matrix fraction

$$c(sI - a)^{-1}b = D^{-1}(s)N_o(s)$$

is constructed (see **bhf2rmf**). N follows from

$$N(s) = N_o(s) + D(s)d(s)$$

This algorithm also applies if the variable is p , z or q . If the indeterminate variable is z^{-1} or d then a final conversion of the fraction with the help of **reverse** is needed.

The algorithm for **ss2rmf** is dual to that of **ss2lmf**.

Diagnostics

Error messages are issued under the following conditions:

- The input matrices a , b and c are not constant matrices
- The input matrices have incompatible sizes

See also

lmf2ss , rmf2ss	conversion of to a left or right polynomial matrix fraction to state space representation
dss2rmf , rmf2dss dss2lmf , lmf2dss	conversion from descriptor representation to right or left matrix fraction or vice-versa
ss	create state space models or convert LTI model to state space form

stab

Purpose Stabilizing controllers for linear systems

Syntax

```
[Nc,Dc] = stab(N,D)
[Nc,Dc] = stab(N,D,'l')
[Nc,Dc] = stab(N,D,'r')
[Nc,Dc,E,F] = stab(N,D)
[Nc,Dc,E,F] = stab(N,D,'l')
[Nc,Dc,E,F] = stab(N,D,'r')
```

Description Given a linear time-invariant plant transfer matrix

$$P(v) = D^{-1}(v)N(v)$$

where v can be any of s , p , z , q , z^{-1} and d , the commands

```
[Nc,Dc] = stab(N,D)
[Nc,Dc]=stab(N,D,'l')
```

compute a stabilizing controller as in Fig. 19 with transfer matrix

$$Q(v) = N_C(v)D_C^{-1}(v) .$$

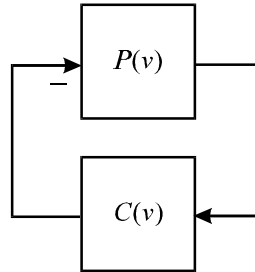


Fig. 19. Feedback structure

The resulting closed-loop poles are randomly placed in the stability region (whose shape naturally depends on the particular choice of operator). For the same plant, the commands

$$[N_c, D_c, E, F] = \text{stab}(N, D)$$

$$[N_c, D_c, E, F] = \text{stab}(N, D, 'l')$$

are used to get the parametrization of all stabilizing controllers in the form

$$C(v) = (N_c(v)P(v) + E(v)T(v))(D_c(v)P(v) - F(v)T(v))^{-1}$$

$P(v)$ is an arbitrary but stable polynomial matrix parameter of compatible size and $T(v)$ is another (not necessarily stable) arbitrary polynomial matrix parameter of compatible size. The parameter can be chosen at will but so that the resulting controller is proper (or causal) (Kucera 1979; Kucera, 1991).

Notice: If any common factor that appear in $Q(v)$ is cancelled, then the above formula is the standard (Youla-Kucera) parametrization of all stabilizing controllers, and $\det P(v)$ is the resulting closed-loop characteristic polynomial.

Similarly, for a plant transfer matrix

$$P(v) = N(v)D^{-1}(v)$$

the command

$$[\mathbf{Nc}, \mathbf{Dc}] = \text{stab}(\mathbf{N}, \mathbf{D}, 'r')$$

computes a stabilizing controller with transfer matrix described by

$$Q(v) = D_C^{-1}(v)N_C(v) .$$

and the command

$$[\mathbf{Nc}, \mathbf{Dc}, \mathbf{E}, \mathbf{F}, \text{degT}] = \text{stab}(\mathbf{N}, \mathbf{D}, 'r')$$

gives rise to the parametrization

$$Q(v) = (P(v)D_c(v) - T(v)F(v))^{-1}(P(v)N_C(v) + T(v)E(v))$$

$P(v)$ is an arbitrary but stable polynomial matrix parameter of compatible size and $T(v)$ is another (not necessarily stable) arbitrary polynomial matrix parameter of compatible size. The parameter can be chosen at will but so that the resulting controller is proper (or causal).

Note: If the plant is strictly proper (or causal) then the resulting controller is always proper (causal). Otherwise its properness (causality) is not guaranteed and should be checked separately.

Note: The macro handles only coprime polynomial matrix fraction descriptions of the plant. If the fraction has a stable factor in common then the user should cancel it before running **stab**.

Example 1

Consider a simple continuous-time plant with

```
d = 2-3*s+s^2,n=s+1
```

```
d =
```

```
2 - 3s + s^2
```

```
n =
```

```
1 + s
```

The plant has two unstable poles

```
roots(d)
```

```
ans =
```

```
2.0000
```

```
1.0000
```

To obtain a stabilizing controller, type

```
[nc,dc] = stab(n,d)
```

```
nc =
```

```
73 - 2.8s
```

```
dc =
```

```
9.9 + s
```

Indeed, this controller gives rise to the closed-loop characteristic polynomial

```
cl = d*dc+n*nc
```

```
cl =
```

```
93 + 43s + 4s^2 + s^3
```

and all closed-loop poles are in the left half plane

```
roots(cl)
```

```
ans =
```

```
-0.8297 + 6.1991i
```

```
-0.8297 - 6.1991i
```

```
-2.3816
```

Using the polynomial approach, not only one but all stabilizing controllers are obtained. Type

```
[nc,dc,e,f] = stab(n,d)
```

```
nc =
```

```
1.4 + 6.8s
```

```
dc =
```

```
0.98 + s
```

```
e =
```

```
2 - 3s + s^2
```

```
f =
```

```
1 + s
```

to get another (because of the macro random character) stabilizing controller along with the whole parameterization

$$\frac{n_{\text{controller}}}{d_{\text{controller}}} = \frac{(1.396 + 6.83s)c(s) + (2 - 3s + s^2)t(s)}{(0.9801 + s)c(s) - (1 + s)t(s)}$$

Taking $c(s) = 1$ and $t(s)$ arbitrary we obtain all the controllers that assign the closed-loop characteristic polynomial to be

```
m = d*dc+n*nc
```

```
m =
```

```
3.4 + 7.3s + 4.8s^2 + s^3
```

with closed-loop poles at

```
roots(m)
```

```
ans =
```

```
-2.1221
```

```
-1.8176
```

```
-0.8701
```

Similarly, for a fixed stable $c(s)$ and arbitrary $t(s)$, unless we make any cancellation in the controller, we always obtain the closed-loop characteristic polynomial $m(s)c(s)$. If we choose a $t(s)$ for which $m(s)$ cancels, however, and perform the cancellation then the resulting closed-loop characteristic polynomial equals exactly $c(s)$. So for

```
c = (s+1)*(s+2)*(s+3)
```

```
c =
```

```
6 + 11s + 6s^2 + s^3
```

and

`t`

`t =`

`2.525 + 3.616s + 1.17s^2`

we obtain

`nc1 = nc*c+e*t`

`nc1 =`

`13.42 + 55.99s + 77.52s^2 + 42.48s^3 + 8s^4`

`dc1 = dc*c-f*t =`

`dc1 =`

`3.356 + 10.64s + 12.09s^2 + 5.81s^3 + s^4`

Both polynomials are divisible by m . Indeed,

`nc2 = nc1/m`

`nc2 =`

`4 + 8s`

`dc2 = dc1/m`

`dc2 =`

$$1 + s$$

Applying this controller, we get

```

c12 = d*dc2+n*nc2

c12 =

      6 + 11s + 6s^2 + s^3

roots(c12)

ans =

-3.0000

-2.0000

-1.0000

```

This equals the desired $c(s)$.

Example 2

Consider the three-input two-output discrete-time plant given by the transfer matrix $P(z^{-1}) = N(z^{-1})D^{-1}(z^{-1})$ with

```

N = [2 zi zi+1; 1-2*zi 0 zi]

N =

      2              z^-1      1 + z^-1

```

$$1 - 2z^{-1} \quad 0 \quad z^{-1}$$

and

```
Dd = diag([2*zi+1 zi-1 1]);
```

```
D = [1 1 1;0 1 zi;0 0 1]*Dd*[1 0 0;zi+1 1 0;zi 1 1]
```

```
D =
```

$$\begin{array}{ccc} 3z^{-1} + z^{-2} & z^{-1} & 1 \\ -1 + 2z^{-2} & -1 + 2z^{-1} & z^{-1} \\ z^{-1} & 1 & 1 \end{array}$$

The plant is stabilized by the controller $Q(v) = D_C^{-1}(v)N_C(v)$ with

```
[Nc,Dc] = stab(N,D,'r')
```

```
Nc =
```

$$\begin{array}{ccc} 16 & -21 + 5.2z^{-1} & \\ 21 & -37 + 7.1z^{-1} & \\ -86 & 1.5e+002 - 29z^{-1} & \end{array}$$

```
Dc =
```

```

-13 + 12z^-1      -2.4 - 6.4z^-1      -2.4 - 4z^-1 +
1.2z^-2

-32 + 13z^-1      5.9 - 5.5z^-1      11 - 3.1z^-1 -
1.6z^-2

1.3e+002 - 57z^-1 -22 + 29z^-1      -22 + 31z^-1

```

Indeed, the feedback matrix

```
C1 = Dc*D+Nc*N
```

```
C1 =
```

```

13 + 11z^-1 + 2.5z^-2      0      0
0      5.1 + 3.1z^-1      0
0      0      19 + 17z^-
1

```

is stable as

```
isstable(C1)
```

```
ans =
```

```
1
```

Algorithm

Given $P = D^{-1}N$, the macro computes $C = N_C D_C^{-1}$ by solving the linear polynomial matrix equation $DD_C + NN_C = R$ with random stable diagonal right hand side matrix

R of suitable degrees. Similarly, given $P = ND^{-1}$, the controller $C = D_C^{-1}N_C$ is constructed by solving the equation $D_C D + N_C N = R$.

Diagnostics

The macro returns error messages in the following situations:

- The input arguments are not polynomial objects
- The input matrices have inconsistent dimensions
- The input string is incorrect
- The input polynomial matrices are not coprime

See also

`pplace` pole placement
`debe` deadbeat regulator design

stabint

Purpose Robust stability interval for a single parameter uncertain polynomial

Syntax

```
[rmin,rmax] = stabint(p0,p1,...,pn)
[rmin,rmax] = stabint(p0,p1,...,pn,tol)
```

Description Given an uncertain polynomial

$$p(s,r) = p_0(s) + rp_1(s) + r^2p_2(s) + \dots + r^np_n(s)$$

with a single real scalar uncertain parameter r that is nominally stable (i.e., such that $p(s,0) = p_0(s)$ is stable), the command

```
[rmin,rmax] = stabint(p0,p1,...,pn)
```

returns the smallest negative real number r_{\min} and the largest positive real number r_{\max} such that the polynomial $p(s,r)$ remains stable for all $r \in (r_{\min}, r_{\max})$. In other words, (r_{\min}, r_{\max}) is the largest robust stability interval that contains $r = 0$ for the uncertain polynomial $p(s,r)$. The macro works both for continuous- and discrete-time polynomials.

Note: To avoid possible degree drop problems it is required that $\deg p_0 > \deg p_i$ for all $i = 1, 2, \dots, n$.

If $P(s,r) = P_0(s) + rP_1(s) + r^2P_2(s) + \dots + r^nP_n(s)$ is a square polynomial matrix depending on a single uncertain parameter then the command

```
[rmin,rmax] = stabint(P0,P1,...,Pn)
```

finds the largest robust stability interval for its (2-D) determinant

$$\det P(s,r) = q(s,r) = q_0(s) + rq_1(s) + \dots + r^mq_m(s)$$

where $q_0(s) = \det P_0(s)$, and, of course, $m \geq n$. However, then it is necessary to have $\deg q_0 > \deg q_i$ for all $i = 1, \dots, m$. That is, the condition must hold for degrees in the determinant and not only for the degrees of the given matrices P_i .

Note: The macro may be called with a local tolerance as the last input argument. On the other hand, the last input argument is considered to be the tolerance whenever it is a scalar.

Example 1

The call

```
[rmin,rmax]=stabint(1+s,pol(1))
```

returns the evident answer

```
rmin =  
-1  
rmax =  
Inf
```

and so does the call

```
[rmin,rmax]=stabint(z,pol(1))

rmin =

    -1

rmax =

     1
```

Note that here $p_n = 1$ must be put in as `pol(1)`. If just 1 is typed then it is mistakenly considered to be a desired tolerance value.

Example 2

To check for robust stability interval the uncertain polynomial

$$\rho(s,r) = (3-r) + (2-q)s + s^2,$$

write it first as

$$\rho(s,r) = (3 + 2s + s^2) + r(-1 - s)$$

and input the data

```
p0=3+2*s+s^2,p1=-1-s

p0 =

    3 + 2s + s^2
```

```
p1 =  
      -1 - s
```

Then test nominal stability by

```
isstable(p0)  
  
ans =  
  
      1
```

and finally call

```
[rmin,rmax]=stabint(p0,p1)  
  
rmin =  
  
      -Inf  
  
rmax =  
  
      2.0000
```

This result may be confirmed by the root-locus-like plot of Fig. 20 for a fictitious plant $p_1(s)/p_0(s)$:

```
rlocus(ss(-s-1,s^2+2*s+3),-10:.01:2)
```

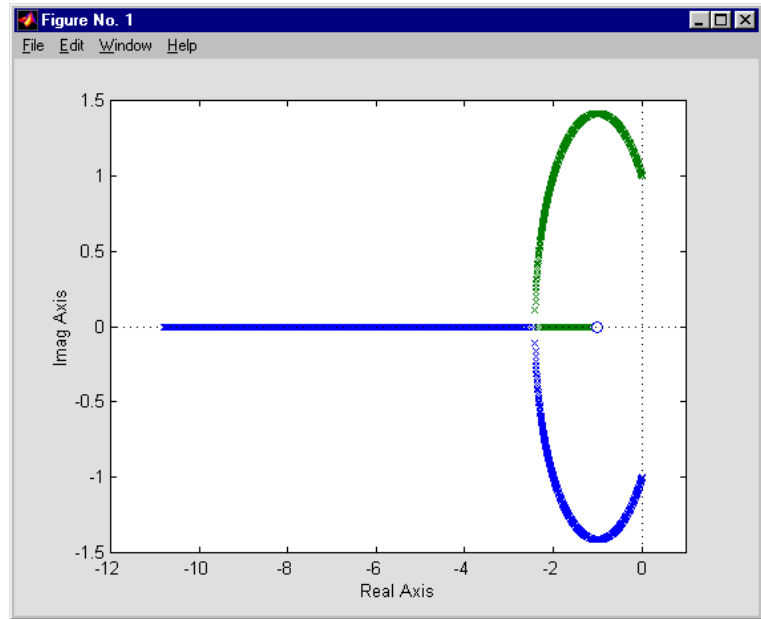



Fig. 20. Root locus plot

Example 3

Finally consider the single parameter uncertain matrix

$$P(s,r) = \begin{bmatrix} 1+r+2s+s^2 & -1-s \\ 1+s & s+rs+s^2 \end{bmatrix}$$

Typing

```
P0=[1+2*s+s^2,-1-s;1+s,s+s^2]
```

```
P0 =
```

$$\begin{array}{cc} 1 + 2s + s^2 & -1 - s \\ 1 + s & s + s^2 \end{array}$$

```
P1=[1 0;0 s]
```

```
P1 =
```

$$\begin{array}{cc} 1 & 0 \\ 0 & s \end{array}$$

and

```
[rmin,rmax]=stabint(P0,P1)
```

we obtain the interval

```
rmin =
```

```
-0.6633
```

```
rmax =
```

```
Inf
```

Algorithm The method of the guardian map is used as described by Barmish (1996). The matrix case is reduced to scalars by computing a two-dimensional determinant.

Diagnostics The macro returns error messages in the following situations:

- The input arguments are not polynomial objects
- The input matrices have inconsistent dimensions
- There are less than two input polynomials
- The first input argument is not a stable polynomial
- The degree condition is not satisfied

See also

<code>ptopex</code>	extreme polynomials for polytope of polynomials
<code>ptopplot</code>	plot value set of a polytope of polynomials

subsasgn

Purpose Subscripted assignment for a polynomial matrix

Syntax $P\{pwr\} = R$

$P(ri,ci) = R$

$P.var = string$

$P.user = string$

Description If P is the polynomial matrix

$$P = P_0 + P_1*s + P_2*s^2 + \dots + P_d*s^d$$

then the command

$$P\{0:d\} = R$$

sets the compound coefficient matrix

$$[P_0 \ P_1 \ P_2 \ \dots \ P_d]$$

equal to R . The assignment

$$P\{k\} = R$$

sets the k th coefficient matrix P_k of P equal to R . The command

`P(ri,ci) = R`

sets the polynomial submatrix of P of row indices `ri` and column indices `ci` equal to R .

The assignment

`P.var = R`

sets `v`, the indeterminate string variable of P , while the command

`P.user = R`

sets `u`, the user data of P .

The subscripted references may be combined. For instance,

`P{0}(ri,ci)= R`

sets the constant coefficient matrix corresponding to a submatrix of P equal to R .

Examples

Let

`P = [1+2*s 3+4*s^2]`

`P =`

`1 + 2s 3 + 4s^2`

We may modify the constant coefficient matrix of P by a command such as

```
P{0} = [5 6]
```

This results in

```
P =
      5 + 2s      6 + 4s^2
```

The (1,2) entry may be changed according to

```
P(1,2) = 7*s^3
```

```
P =
      5 + 2s      7s^3
```

The indeterminate variable may be redefined by typing

```
P.var = 'z'
```

```
P =
      5 + 2z      7z^3
```

Algorithm

The macro employs basic MATLAB 5 operations.

Diagnostics

The macro returns error messages under numerous erroneous input conditions. It issues a warning if inconsistent variables are encountered.

See also

suboref subscripted reference for a polynomial matrix

subsref

Purpose Subscripted reference for a polynomial matrix

Syntax

`P{pwr}`

`P(ri,ci)`

`P.coef`

`P.deg`

`P.size`

`P.user`

`P.var`

Description If P is the polynomial matrix

$$P = P_0 + P_1*s + P_2*s^2 + \dots + P_d*s^d$$

then

`P{di}`

returns the block row constant matrix corresponding to the degree indices `di`. In particular,

`P{0:d}`

is `[P0 P1 P2 .. Pd]`,

`P{0}`

is `P0`, the constant coefficient matrix of P , and

`P{k}`

is `Pk`, the k th coefficient matrix of P . The call

`P(ri,ci)`

returns the polynomial submatrix of P with row indices `ri` and column indices `ci`.

The call

`P.deg`

returns the degree of P ,

`P.size`

returns the size of P ,

`P.coef`

returns the three dimensional coefficient matrix of P ,

`P.var`

returns the indeterminate string variable of P , and

`P.user`

returns the user data of P .

The subscripted references may be combined. For instance, `P(ri,ci).deg` returns the degree of a submatrix of P , and `P{di}(ri,ci)` returns the coefficient matrix corresponding to the degree `di` of the submatrix `P(ri,ci)`.

Examples

Let

```
P = [1+2*s 3+4*s^2]
```

```
P =
```

```
1 + 2s      3 + 4s^2
```

The constant coefficient matrix follows as

```
P{0}
```

```
ans =
```

```
1      3
```

while the (1,2) entry may be retrieved according to

```
P(1,2)
```

```
ans =
```

```
3 + 4s^2
```

The degree of the (1,1) entry is

```
P(1,1).deg
```

```
ans =
```

```
1
```

Algorithm

The macro employs basic MATLAB 5 operations.

Diagnostics

The macro returns error messages under numerous erroneous input conditions.

See also

subsasgn subscripted assignment for a polynomial matrix

sylv

Purpose Create the Sylvester matrix of a polynomial matrix

Syntax

```
S = sylv(A)
```

```
S = sylv(A,k)
```

```
S = sylv(A,'row')
```

```
S = sylv(A,'col')
```

```
S = sylv(A,k,'row')
```

```
S = sylv(A,k,'col')
```

Description

The commands

```
S = sylv(A,k)
```

```
S = sylv(A,k,'row')
```

create the Sylvester resultant matrix S of order k from the polynomial matrix A . If

$$A(s) = A_0 + A_1 s + A_2 s^2 + \dots + A_d s^d$$

then

$$S = \begin{bmatrix} A_0 & A_1 & \dots & A_d & 0 & \dots & \dots & 0 \\ 0 & A_0 & A_1 & \dots & A_d & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & 0 & A_0 & A_1 & \dots & A_d \end{bmatrix}$$

Block row 1
Block row 2
...
Block row $k+1$

The integer k is the number of zero blocks in each block row. Its default value is $k = d$. In the form

`S = sylv(A,k,'col')`

a column Sylvester matrix is formed.

Examples

If

`A = 1+2*s+3*s^2;`

then

`sylv(A)`

returns

`ans =`

1	2	3	0	0
0	1	2	3	0
0	0	1	2	3

while

```
sylv(A,4,'col')
```

results in

```
ans =
```

1	0	0	0	0
2	1	0	0	0
3	2	1	0	0
0	3	2	1	0
0	0	3	2	1
0	0	0	3	2
0	0	0	0	3

Algorithm

The macro uses standard MATLAB 5 and Polynomial Toolbox operations.

Diagnostics

The macro issues an error message if

- It does not have enough input arguments
- It has too many numerical or string inputs
- The requested order is not a nonnegative integer

- An incorrect input string is encountered

See also

`axb`, `axbc`, `axybc` linear polynomial matrix equation solvers
`axxab`, `axyab`, `axybc`
`xaaxb`, `xab`, `xaybc`

sym

Purpose

Convert a polynomial matrix to the format of the Symbolic Toolbox

Syntax

```
PS = sym(P)
```

```
PS = sym(P,var)
```

Description

The commands

```
PS = sym(P)
```

```
PS = sym(P,var)
```

convert a polynomial matrix P in the Polynomial Toolbox format to a matrix \mathbf{PS} in the format of the Symbolic Toolbox.

The entries of \mathbf{PS} are polynomial expressions in the indeterminate \mathbf{var} . If \mathbf{var} is not specified then the variable of P is copied.

Examples

Given the polynomial matrix defined in Polynomial Toolbox format by

```
P = [ 1+2s      0
      1      3s^2 ];
```

the command

```
sym(P)
```

returns

```
ans =  
[ 2*s+1,      0 ]  
[      1, 3*s^2 ]
```

The command

```
sym(P,'d')
```

on the other hand, results in

```
ans =  
[ 2*d+1,      0]  
[      1, 3*d^2]
```

Algorithm

The macro uses standard MATLAB 5, Polynomial Toolbox and Symbolic Toolbox commands. It requires the directory of the Symbolic Toolbox to be in the MATLAB search path.

Diagnostics

The macro returns an error message if the input is inappropriate.

See also

`pol`, `lop` specify a polynomial matrix or convert a matrix in standard MATLAB or Symbolic Toolbox format to Polynomial Toolbox format

tf

Purpose	Create transfer function models or convert an LTI system to a transfer function model
Syntax	<pre>sys = tf(N,D) sys = tf(N,D,'l') sys = tf(N,D,'r') sys = tf(N,D,T[,tol]) sys = tf(N,D,'l',T[,tol]) sys = tf(N,D,'r',T[,tol])</pre>
Description	<p>The command <code>tf</code> is a function from the MATLAB Control System Toolbox that creates transfer function objects, or converts other LTI objects to transfer function objects. A model is in transfer function form if the numerator polynomial n_{ij} and the denominator d_{ij} of each entry</p>

$$H_{ij} = \frac{n_{ij}}{d_{ij}}$$

of the transfer matrix H of the system are specified.

The function `tf` has been overloaded so that in addition to its Control System Toolbox functionality it converts systems represented in left or right polynomial matrix fraction form to a Control System Toolbox LTI object in transfer function form.

Let D be a square nonsingular row reduced polynomial matrix and N a polynomial matrix with the same number of rows. Then the commands

```
sys = tf(N,D)
```

```
sys = tf(N,D,'l')
```

convert the system with transfer matrix

$$H = D^{-1}N$$

to a Control System Toolbox LTI object in transfer function form.

If N and D are polynomial matrices in s or p then a continuous-time LTI object is created. If they are matrices in z , q , z^{-1} or d then a discrete-time object with sampling time $T = 1$ is created. In the latter case the commands

```
sys = tf(N,D,T)
```

```
sys = tf(N,D,'l',T)
```

with T a real number, create a discrete-time system with sampling time T . If $T = 0$ then the object is taken to be a continuous-time system.

If the option '1' is replaced with 'x' then N and D define the system whose transfer matrix is the right fraction

$$H = ND^{-1}$$

D needs to be square nonsingular column reduced with the same number of columns as N .

The optional input parameter `tol` defines a tolerance that is used in the conversion of the matrix fraction to transfer function form. Note that this parameter may only be included if also the input parameter T is included.

The function `tf` only works if the Control System Toolbox is included in the MATLAB path.

Later versions of the Polynomial Toolbox will support a polynomial matrix fraction LTI object.

Examples

If we let

```
N = 1; D = s+1;
```

then we obtain

```
tfsys = tf(N,D)
```

```
Transfer function:
```

```
1
```

```
-----
```

```
s + 1
```

On the other hand,

```
N = [ 1
```

```
1 ];
```

```
D = [ 1+z  0
```

```
2  z^3 ];
```

results in

```
tfsys = tf(N,D)
```

```
Transfer function from input to output...
```

```
1
```

```
#1: -----
```

```
z + 1
```

```
z - 1
```

```
#2: -----
```

```
z^4 + z^3
```

Sampling time: 1

Algorithm

The function calls `lmf2tf` or `rmf2tf`.

Diagnostics

The function returns an error message in the following circumstances:

- The Control System Toolbox is not included in the MATLAB path
- N and D are polynomial matrices in different variables
- The third, fourth or fifth input argument are incorrect

See also

<code>lti2lmf</code> , <code>lti2rmf</code>	conversion of a state space LTI object to a left or right matrix fraction
<code>ss</code>	creation of a transfer function LTI object or conversion to a state space model
<code>zpk</code>	creation of a zero-pole-gain LTI object or conversion to a zero-pole-gain model
<code>lmf2tf</code> , <code>rmf2tf</code> <code>tf2lmf</code> , <code>tf2rmf</code>	conversion from a left or right polynomial matrix fraction model to a transfer function model, and vice-versa

transpose (.)

Purpose Transpose of a polynomial matrix

Syntax `At = A.'`

`At = transpose(A)`

Description For a polynomial matrix A , the commands

`At = A.'`

`At = transpose(A)`

return the polynomial matrix `At` that is formed by transposing the matrix A .

Examples Consider

`P = [(1+i)*s (1-i)*s+s^2];`

The conjugate transpose of this polynomial matrix is

`P.'`

`ans =`

`(1+1i)s`

`(1-1i)s + s^2`

If

```
P = [(1+i)*z (1-i)*z+z^2];
```

then

```
P.'
```

similarly results in

```
ans =  
  
(1+1i)z  
  
(1-1i)z + z^2
```

Algorithm

The macro **transpose** uses standard MATLAB operations.

Diagnostics

The macro displays an error message if it has too many input arguments or too many output arguments.

See also

conj	conjugate of a polynomial matrix
ctranspose (')	complex conjugate transpose of a polynomial matrix
real	real part of a polynomial matrix
imag	imaginary part of a polynomial matrix

tri**Purpose**

Triangular or staircase form of a polynomial matrix

Syntax

```
[T,U,rowind] = tri(A)
[T,U,rowind] = tri(A,'col')
[T,U,colind] = tri(A,'row')
[T,U,rowind] = tri(A,degree)
[T,U,rowind] = tri(A,degree,'col')
[T,U,colind] = tri(A,degree,'row')
[T,U,ind] = ...
    tri(A[,degree][, 'col'|, 'row'][, 'syl'|, 'gau'][, tol])
```

Description

Given an arbitrary polynomial matrix A , the commands

```
[T,U,rowind] = tri(A)
[T,U,rowind] = tri(A,'col')
```

compute a column staircase form T of A . In particular, if A is non-singular then T is a lower-left triangular matrix. The unimodular reduction matrix U is such that $AU = T$, and the i th entry `rowind(i)` of the row vector `rowind` is the row index of the

uppermost non-zero entry in the i th column of T . Note that `rowind(i)` = 0 if the i th column of T is zero. As a result, the number of non-zero elements in `rowind` is the algorithm's idea of the rank of A . The off-diagonal elements in the staircase form are not necessarily reduced, so the output generally differs from that of macro `hermite`.

Similarly,

```
[T,U,colind] = tri(A,'row')
```

computes a row staircase form of A , the unimodular reduction matrix U such that $UA = T$ and a column vector `colind` such that `colind(i)` is the column index of the leftmost non-zero entry in the i th row of T .

The command

```
tri(A,degree)
```

seeks a reduction matrix U of given degree `degree`. If `degree` is not specified then a reduction matrix of minimum overall degree is computed by an iterative scheme. If `degree` is negative then a reduction matrix of maximum achievable degree is returned.

The command

```
tri(A,'syl')
```

performs the transformation through stable reductions of Sylvester matrices. This method is preferable numerically and is the default method. The command

`tri(A,'gau')`

performs the transformation through a modified version of Gaussian elimination. This method is preferable esthetically.

An optional tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Consider the polynomial matrix

```
A = [ s^2    0    1
      0    s^2  1+s];
```

To find its column staircase form and associated quantities we type

```
[T,U,rowind] = tri(A)
```

MATLAB returns

`T =`

```
-1      0      0
-1 - s  -1.2s^2  0
```

`U =`

```
0      0.29      0.5
0  -0.87 + 0.29s  0.5 + 0.5s
```

```

-1      -0.29s^2      -0.5s^2
rowind =
1      2      0

```

Next consider

```

B = [ 1-d^2    0    1-d
      0    1+d    0
      1-d^2  1+d    1  ];

```

By the command

```
T = tri(B)
```

we obtain the lower triangular form

```

T =
-1.2 + 1.2d      0      0
0      -1 - d      0
-1.2 + 0.41d + 0.41d^2  -1 - d  -0.58d - 0.58d^2

```

Note that this matrix is not in column Hermite form.

Algorithm

The command `tri(A, 'syl')` performs the reduction to triangular form through *QR* decomposition of a Sylvester matrix of given order d . The *QR* decomposition is performed by Householder transformations in the macro `cef`. If the reduction of a Sylvester matrix of order d is successful then it results in a unimodular matrix of degree d . If the reduction fails then the order d must be increased.

The critical point is how to find the minimum value of d such that the reduction is successful. In order to find this value, the macro performs a simple bisection (dichotomy) search. Guaranteed bounds on d are `d_lower` = 0 and

$$\begin{aligned} d_{\text{upper}} = \min(&cd(1) + cd(2) + \dots + cd(nc-1), \\ &rd(1) + rd(2) + \dots + rd(nr-1)) \end{aligned}$$

where `cd(1) ≥ cd(2) ≥ ... ≥ cd(nc-1) ≥ cd(nc)` are column degrees and `rd(1) ≥ rd(2) ≥ ... ≥ rd(nr-1) ≥ rd(nr)` are row degrees of the $nc \times nr$ input matrix A .

To bypass the dichotomy search on d the user may provide the reduction degree in the form of the command `tri(A, d)`. If the reduction with order d fails then `d_low` is set equal to d and the dichotomy search is carried out

Another method to bypass the dichotomy search on d is to use the syntax `tri(A, -1)`. Then we d is set equal to `d_up` and the triangularization is performed only once.

Note however that, depending on the input matrix, several triangularizations of a Sylvester matrix of low order may be computationally less intensive than one triangularization of a Sylvester matrix of maximum order $d = \mathbf{d_up}$.

If the unimodular reduction matrix is required then it is computed by reducing the compound matrix $[A; I]$ to triangular form, so that

$$[A; I] * U = [T; U]$$

provides both the triangular form T and the unimodular matrix U .

All the above points are described in Henrion and Sebek (1998b, 1999).

The command `tri(A, 'gau')` performs the reduction of Sylvester matrices of increasing orders by a modified version of Gaussian elimination with partial pivoting. The algorithm is designed to preserve the shift-invariant Toeplitz structure of the Sylvester matrix (Labhalla, Lombardi and Marlin, 1996). It is actually a specialization to triangularization of the greatest common divisor extraction algorithm described in Bitmead *et al.* (1978).

Diagnostics

The macro `tri` issues error messages if

- An invalid input argument or matrix argument is encountered
- An invalid option is encountered
- Invalid **Not-a-Number** or **Infinite** entries are found in the input matrices
- The reduction fails in one of several ways

In the latter case the tolerance should be modified. A warning follows if the user-supplied degree is greater than the maximum expected degree.

See also

hermite Hermite form of a polynomial matrix

smith Smith form of a polynomial matrix

tril

Purpose Extract the lower triangular part of a polynomial matrix

Syntax

```
Q = tril(P)
Q = tril(P,k)
```

Description The command

```
Q = tril(P)
```

returns a matrix Q whose lower triangular part (including the diagonal) equals that of the polynomial matrix P . The strictly upper triangular entries are zero.

The command

```
Q = tril(P,k)
```

returns a lower triangular matrix that retains the polynomial elements of P on and below the k -th diagonal and sets the remaining elements equal to 0. The values $k = 0$, $k > 0$ and $k < 0$ correspond to the main, superdiagonals and subdiagonals, respectively.

Examples The commands

```
P = [ 1+s      2*s^2
      -2      3+4*s^3 ];
```

```
Q = tril(P)
```

produce the response

```
Q =
      1 + s      0
     -2      3 + 4s^3
```

while

```
Q = tril(P,-1)
```

results in

```
Constant polynomial matrix: 2-by-2
Q =
      0      0
     -2      0
```

Algorithm

The routine uses standard routines from the Polynomial Toolbox.

Diagnostics

The macro displays no error messages.

See also

<code>triu</code>	extract the upper triangular part
<code>diag</code>	extract diagonals or create diagonal matrices

triu

Purpose	Extract the upper triangular part of a polynomial matrix
Syntax	$Q = \text{triu}(P)$ $Q = \text{triu}(P,k)$
Description	<p>The command</p> $Q = \text{triu}(P)$ <p>returns a matrix Q whose upper triangular part (including the diagonal) equals that of the polynomial matrix P. The strictly upper triangular entries are zero.</p> <p>The command</p> $Q = \text{triu}(P,k)$ <p>returns a upper triangular matrix that retains the polynomial elements of P on and above the k-th diagonal and sets the remaining elements equal to 0. The values $k = 0$, $k > 0$ and $k < 0$ correspond to the main, superdiagonals and subdiagonals, respectively.</p>

Examples The commands

```
P = [ 1+s      2*s^2
      -2      3+4*s^3 ];
```

```
Q = triu(P)
```

produce the response

```
Q =
      1 + s      2s^2
      0          3 + 4s^3
```

while

```
Q = triu(P,1)
```

results in

```
Q =
      0      2s^2
      0      0
```

Algorithm

The routine uses standard routines from the Polynomial Toolbox.

Diagnostics

The macro displays no error messages.

See also

tril extract the lower triangular part
diag extract diagonals or create diagonal matrices

zpk

Purpose Create zero-pole-gain models or convert an LTI system to a zero-pole-gain model

Syntax

```
sys = zpk(N,D)

sys = zpk(N,D,'l')

sys = zpk(N,D,'r')

sys = zpk(N,D,T[,tol])

sys = zpk(N,D,'l',T[,tol])

sys = zpk(N,D,'r',T[,tol])
```

Description

The command **zpk** is a function from the MATLAB Control System Toolbox that creates zero-pole-gain (ZPK) objects, or converts other LTI objects to zero-pole-gain objects. A model is in zero-pole-gain form if each entry H_{ij} of the transfer matrix of the system is specified in the form

$$H_{ij}(s) = k \frac{\prod_j (s - z_j)}{\prod_j (s - p_j)}$$

with $z_j, j = 1, 2, \dots$, its zeros, $p_j, j = 1, 2, \dots$, its poles, and k its gain.

The function **zpk** has been overloaded so that in addition to its Control System Toolbox functionality it converts systems represented in left or right polynomial matrix fraction form to a Control System Toolbox LTI object in zero-pole-gain form.

Let D be a square nonsingular row reduced polynomial matrix and N a polynomial matrix with the same number of rows. Then the commands

```
sys = zpk(N,D)
```

```
sys = zpk(N,D, 'l')
```

convert the system with transfer matrix

$$H = D^{-1}N$$

to a Control System Toolbox LTI object in zero-pole-gain form.

If N and D are polynomial matrices in s or p then a continuous-time LTI object is created. If they are matrices in z , q , z^{-1} or d then a discrete-time object with sampling time $T = 1$ is created. In the latter case the commands

```
sys = zpk(N,D,T)
```

```
sys = zpk(N,D, 'l', T)
```

with T a real number, create a discrete-time system with sampling time T . If $T = 0$ then the object is taken to be a continuous-time system.

If the option 'l' is replaced with 'r' then N and D define the system whose transfer matrix is the right fraction

$$H = ND^{-1}$$

D needs to be square nonsingular column reduced with the same number of columns as N .

The optional input parameter `tol` defines a tolerance that is used in the conversion of the matrix fraction to zero-pole-gain form. Note that this parameter may only be included if also the input parameter T is included.

The function `zpk` only works if the Control System Toolbox is included in the MATLAB path.

Later versions of the Polynomial Toolbox will support a polynomial matrix fraction LTI object.

Examples

The command

```
zpksys = zpk(s+1,s+2)
```

results in

```
Zero/pole/gain:
(s+1)
-----
(s+2)
```

On the other hand

```

N = [ 1;
      1 ];
D = [ z+1  2
      0   z^2 ];

```

results in

```
zpksys = zpk(N,D)
```

Zero/pole/gain from input to output...

(z-1.414) (z+1.414)

#1: -----

z^2 (z+1)

1

#2: ---

z^2

Sampling time: 1

Algorithm

The function calls `lmf2zpk` or `rmf2zpk`.

Diagnostics

The function returns an error message in the following circumstances

- The Control System Toolbox is not included in the MATLAB path
- N and D are polynomial matrices in different variables
- The third, fourth or fifth input argument are incorrect.

See also

<code>lti2lmf</code> , <code>lti2rmf</code>	conversion of a state space LTI object to a left or right matrix fraction
<code>ss</code>	creation of a transfer function LTI object or conversion to a state space model
<code>tf</code>	creation of a transfer function LTI object or conversion to a transfer function model
<code>lmf2zpk</code> , <code>rmf2zpk</code> <code>zpk2lmf</code> , <code>zpk2rmf</code>	conversion from a left or right polynomial matrix fraction model to a zero-pole-gain model, and vice-versa

zpk2lmf, zpk2rmf

Purpose	Conversion of a transfer matrix in zero-pole-gain (ZPK) format to a left or right matrix fraction
Syntax	$[N,D] = \text{zpk2lmf}(Z,P,K[,tol])$ $[N,D] = \text{zpk2rmf}(Z,P,K[,tol])$
Description	Given two cell arrays Z and P and a matrix K that together define a transfer matrix H in ZPK format, the function

$$[N,D] = \text{lmf2zpk}(Z,P,K)$$

returns the transfer function as the left coprime fraction

$$H = D^{-1}N$$

In the ZPK format format, each entry

$$H_{ij}(s) = k \frac{\prod_j (s - z_j)}{\prod_j (s - p_j)}$$

of the transfer matrix is characterized by its zeros $z_j, j = 1, 2, \dots$, its poles $p_j, j = 1, 2, \dots$, and its gain k .

Z and P are cell arrays and K is a matrix. Each cell of Z contains the zeros of the corresponding transfer function, each cell of P contains the poles, and each entry of K contains the gain. Note that the polynomials whose gain and roots are represented are taken as polynomials in the default indeterminate variable, and that correspondingly N and D are returned as polynomial matrices in the default variable.

Similarly, the function

$$[N,D] = \text{zpk2rmf}(Z,P,K)$$

returns the transfer function as the right coprime fraction

$$H = ND^{-1}$$

In both cases a tolerance `tol` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Examples

Consider the transfer function

$$H(s) = \begin{bmatrix} \frac{2(s-1)}{s+2} & \frac{s(s+3)}{s+2} \end{bmatrix}$$

Inspection shows that its ZPK data are

```
Z = cell(1,2); Z{1,1} = 1; Z{1,2} = [0 -3]
P = cell(1,2); P{1,1} = -2; P{1,2} = -2
```

```

K = [ 2  1 ]

Z =

    [1]    [1x2 double]

P =

    [-2]    [-2]

K =

    2      1

```

The corresponding left matrix fraction is

```

[Nl,Dl] = zpk2lmf(Z,P,K)

Nl =

    -2 + 2s    3s + s^2

Dl =

    2 + s

```

while the right matrix fraction is

```

[Nr,Dr] = zpk2rmf(Z,P,K)

Nr =

```

$$D_r = \frac{0.33 + s}{1} \frac{-2.7}{2.7 + s} \frac{-2}{-2}$$

Algorithm

Using the routine `root2po1` the ZPK data are converted to transfer function format, which is subsequently transformed to matrix fraction form by the routines `tf2lmf` or `tf2rmf`.

Diagnostics

The macros `zpk2lmf` and `zpk2rmf` return error messages if the number of input arguments is incorrect.

See also

<code>lmf2zpk</code> , <code>rmf2zpk</code>	conversion from transfer function format to a left or right polynomial matrix fraction
<code>zpk</code>	create a Control System Toolbox LTI object in ZPK format
<code>tf2lmf</code> , <code>tf2rmf</code> <code>lmf2tf</code> , <code>rmf2tf</code>	conversion of transfer matrices to left or right matrix fractions, and vice-versa

zpplot

Purpose Zero-pole map of a polynomial matrix transfer function

Syntax

```
zpplot(N,D)

zpplot(P)

zpplot(N,D,'new')

zpplot(P,'new')
```

Description Given two polynomial matrices N and D the command

```
zpplot(N,D)
```

computes the roots of D and plots each of them as $*$ and the roots of N and plots each of them as \circ . If the polynomial matrices have consistent dimensions then these roots correspond to the poles and zeros, respectively, of the matrix transfer function $D^{-1}N$ or ND^{-1} .

The command

```
zpplot(P)
```

computes the roots (zeros) of the polynomial matrix P and plots each of them in the complex plane as \circ .

For “discrete-time” matrices (polynomial matrices in the variables z , d or z^{-1}) also the unit circle is plotted.

By including the optional input argument '**new**' a new figure window is opened.

Examples

Let

$$N = 1+2*s+3*s^2; \quad D = 5+6*s+7*s^2+s^3;$$

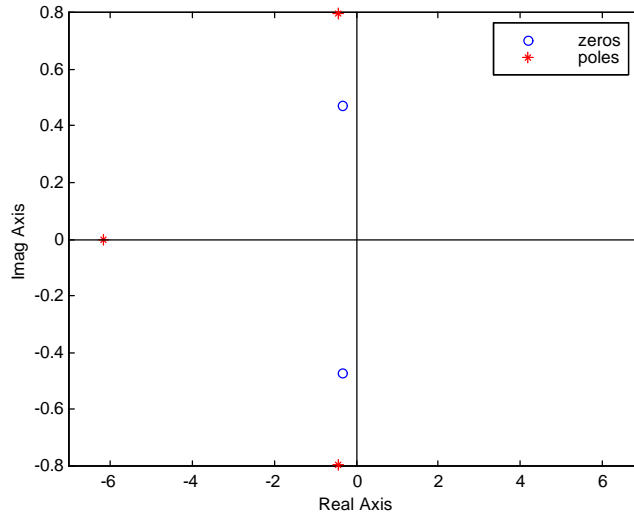


Fig. 21. Pole-zero plot of a transfer function

Then

```
zpplot(N,D)
```

produces the zero-pole plot of Fig. 21. On the other hand, let

```
gensym z, P = prand(4,3,3,'int')
```

```
Polynomial matrix in z: 3-by-3, degree: 4
```

```
P =
```

```
Columns 1 through 2
```

```
-8 + z + z^2 - 6z^3 + 6z^4      6 + 2z^2 + z^3 - z^4
```

```
5 - 4z^3 + z^4                  -7 + 4z + 8z^2 - 3z^3 +  
4z^4
```

```
3 + 4z + 4z^2 + 6z^3 + 3z^4    6 - 6z - z^3 - 8z^4
```

```
Column 3
```

```
4 - 3z + 11z^2 - z^3 + z^4
```

```
6 - 8z - 7z^2 + 3z^3 - 2z^4
```

```
1 - 5z + 7z^2 - 4z^3 + 3z^4
```

The command

zpplot(P)

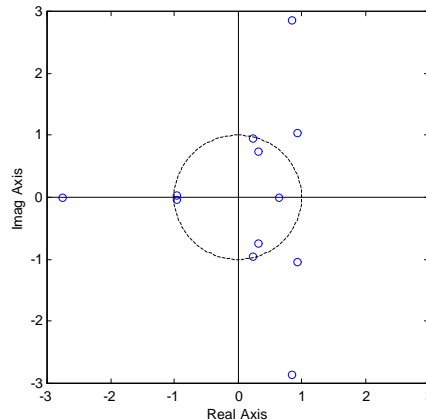
produces Fig. 22.

Algorithm

The macro uses basic operations from MATLAB 5 and the Polynomial Toolbox.

Diagnostics

The macro returns an error message if the number of input or output arguments is incorrect.

See also**roots** Roots of a polynomial matrix**Fig. 22. Plot of the zeros of a “discrete-time” polynomial matrix**